10.01.23

Concurrency

Thread:

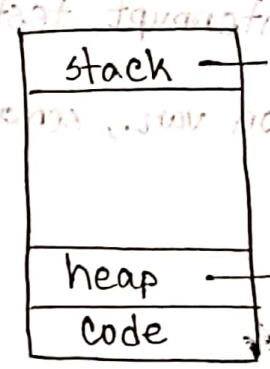
Share an address space

Multi-threaded program structure:

① Producer / Consumer

② Pipeline

③



stores temp var. so individual for every thread.

common for all threads of the same process

14.01.23

*need to work with kernel-level thread.

* Interrupt → timer interrupt
→ syscall / OS-এর কাছে help চাওয়া interrupt

Non-determinism:

*Need mutual-exclusion for critical sections
(shared resource)

low-level primitives: load, store, disable interrupt, test & set
high-level " : monitor, lock, condition var., semapho

Locks:

*ensures mutex

(i) Correctness
— mutex
— progress (deadlock-free)
— bounded (starvation-free)

(ii) Fairness

(iii) Performance

Implementing lock : w/ interrupt

Disadvantage :

① নিজে / other process যেকোনো কাজের জন্যই interrupt call করতে পারবে না।

② only works on uniprocessor.

③ process can keep control of CPU for arbitrary length.

Implementing lock : w/ load+stone

```
lock = 0;

acquire()
{
   while(lock == 1)
   {
   }

   lock = 1;
}
```

① Doesn't ensure mutex. → lock check & set not atomic.

Extra class – IAC.

14.01.23

XCHG: Atomic exchange on test & set

```
int xchg(int *addr, int newval) {
    int old = *addr;
    *addr = newval;
    return old;
}
```

*scheduler will not interrupt until xchg is executed completely.

↙ atomic

Lock implementation with xchg

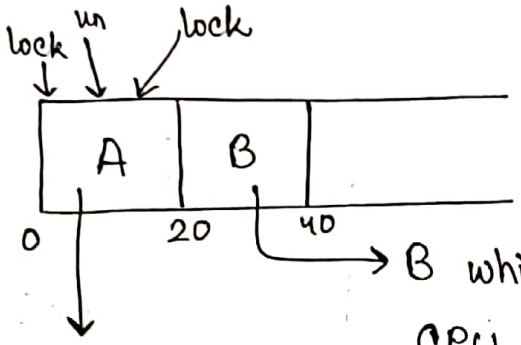
```
void init (lock_t *lock) {
    lock->flag = 0;
}

void acquire (lock_t *lock) {
    while (xchg(lock->flag, 1) == 1)
}

void release(lock_t *lock) {
    lock->flag = 0;
}
```

xchg (&lock → flag, 1) ⇒ returns prev lock value, and assigns flag = 1.

↑ এখানে 1 pass করব, কারণ lock acquire করতে চাই।

Disadvantange:

① Unfair



B while(1) loop-এ অপেক্ষায় থাকবে। (spinlock)

CPU access করছে কিন্তু কাজের কাজ কিছুই হচ্ছে না।

A time slice কোর করল lock acquire করে। আর কোনো process এখন lock পাবে না।

* uniprocessor হলে while(1) check করে লাভ নাই। multiprocessor এর জন্য true.

Fairness: Ticket Lock

	Turn (Ticket)	Ticket
A lock();	0	0
B lock();		1
C lock();		2
A unlock();	1	
A lock();		3
B unlock();	2	
C		

17.01.23

* Ticket lock is a kind of spinlock. Starvation free but wastes CPU

* Slide-এ context switching time = 0 ধরা হয়েছে।
Now, suppose context switching time = 2 sec. Calculate scheduling time.

* কোনো Running process yield করলে state change হয়ে Ready/ Runnable state-এ যাবে।

Spinlock Performance:

Without yield: O(thread * time_slice)

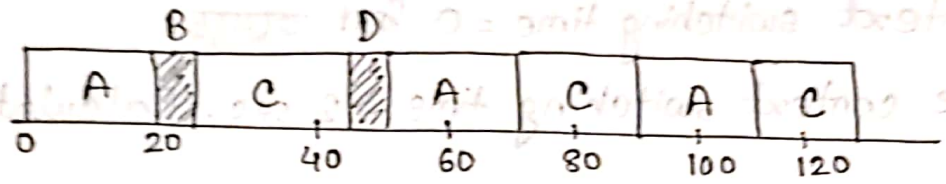
With yield: O(thread * context switch)

→ এটাকে বলে Busy waiting

* যেই process গুলো lock এর জন্য বসে আছে, তাদের CPU time দিবই না, (Minimizing context switch cost) নতুন এটা state assign করব "Waiting" instead of "Runnable".

A, B, D contend for lock, C is not.

* process states: Running, Runnable, Waiting.



* A complete হতে 60ms time লাগবে।

* 4টা process একবার করে Run করার পর —

Waiting : B, D

Runnable : A, C

* A complete হওয়ার পর B এর state Waiting → Runnable এ change করবে। (Waiting এ থাকা শুধু এটা process এর state change করবে।) কিন্তু already A এর পর C Queue তে ছিল; তাই B, C এর পর Queue তে append হবে & A শেষ হওয়ার পর এজন্য C কে Schedule করা হবে।

Lock implementation : Block when waiting

```
typedef struct {
    bool lock = false;
    bool guard = false; // queue access atomic করার জন্য লাগবে
    queue_t queue;      // এবংই lock এর উপর যে যে process
} LockT                    depended, তাদের queue
```

* queue access needs to be atomic.

18.01.23

* yield controls the damage. Can't avoid it.

Block state = sleeping state

mutex = lock

① Why guard needs to be atomic? — to ensure that there's no race condition for the queque

② Why okay to spin on guard? → কারণ time কম লাগবে। Critical section এর উপর spin করলে time waste হবে, processes will be in waiting state.

```
else {
    l->lock = true;
    l->guard = false;
}
```

⇒ Ordering important

আগে guard set করলে CPU নিচে গেলে race cond. arrive করতে পারে। কারণ acquire, release func. while (xchg (&l->guard, true)); check করবে।

③ Why not set lock = false when unpark? → তাহলে আবার প্রতিটা process এর জন্য ~~to~~. acquire lock() call করা লাগবে।

(iv) Is there a race condition? → park() এর আগে unpark() হয়ে যাবে।

* set park() ensures that l→guard = false এর পর park পর্যন্ত যাবে। (Atomic)

Condition Variables

Concurrency objective → Mutex
                      → Ordering

grep "Name" | wc -l // Doesn't ensure আগে grep run করবে।

21.01.23

* thread এর join == process এর wait

Condition Variable :

queue of waiting threads

① wait (cv, ...) ⇒ wait (cond_t *cv, mutex_t *lock)

② signal (cv, ...) ⇒ signal (cond_t *cv)

* child execute শেষ করে exit() call করে zombie state এ যায়, parent-এর কাজ শেষ হলেই (wait() call) child process terminate হবে।
→ Last func. call

Join implementation : Attempt - 1

* Parent-এর wait func. এর ভেতর lock input দিব যাতে parent lock নিয়ে না ঘুমায় যায়। wait func. এর মধ্যে OS unlock করে দিবে।

* Child unlock করার আগে signal দিবে কিন্তু parent উঠে দেখবে lock already acquired → তখন parent wait করবে। এরপর child unlock করে exit করবে, then OS আবার parent কে lock দিবে। parent এরপর বাকি কাজ করবে।

*Need to keep state in addition to CV's!

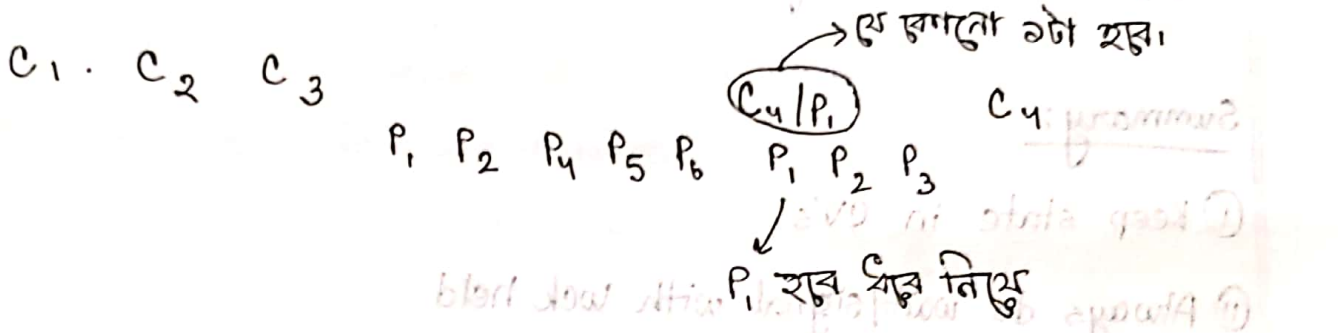
Join Implementation: Attempt-2

Child

```
void thread_exit() {
    done=1;
    Cond_signal(&c);
}
```

⇒ swap করলে deadlock-এ চলে যাবে। কারণ child signal করার পর CPU নিয়ে চলে, parent এসে দেখবে done == 0, ~~তিন~~ এজন্য wait করবে কিন্তু child এরপর done=1 করে exit করে যাবে।

Producer/Consumer Problem:



1 producer, 2 consumer:

P:

P_1 P_2 P_4 P_5 P_6 P_1 P_2 P_3

C_1: C_1 C_2 C_3

C_2: C_1 C_2 C_3 C_4 C_5

* যেহেতু wait, signal same → consumer আরেক consumer কে জাগায় দিতে পারে।

Two CVs:

C1: C_1 C_2 C_3

C2: C_1 C_2 C_4 C_5 C_6

P: P_1 P_2 P_4 P_5 P_6

* checking টা if cond. না দিয়ে while() দিতে হবে।

* How to ensure fairness in this produce/consumer problem?

Summary:

① Keep state in CV's

② Always do wait/signal with lock held

③ Whenever thread wakes from waiting, recheck state

Semaphones

24.01.23

Semaphores

sem_init( )

Wait or Test → decrements value

if ≥ 0 process not sleeping

if < 0 " sleeping

P sem_init(2);

P_1 wait( ); // 1

P_2 wait( ); // 0

P_3 wait( ); // -1 process sleeping

Signal or Post → increments value

Binary Semaphore (Lock):

```
init() { sem = 1; }
acquire() { sem_wait(&lock_t); }
release() { sem_post(&lock_t); }
```

Join with CV vs Semaphones:

```
void thread_join() {
        sem_wait(&s);
}

void thread_exit() {
        sem_post(&s);
}

sem_init(&s, 0);
```

*যতগুলো Dependency ততগুলো semaphores.

Sem-1

Producer/Consume Problem:

*Shared resource/buffer 1টা

Producer

```
while (1) {
    sem_wait(&emptybuffer);
    Fill(&buffer);
    sem_post(&fullbuffer);
```

Consumer

```
while (1) {
    sem_wait(&fullbuffer);
    Use(&buffer);
    sem_post(&emptybuffer);
```

emptybuffer initialize = 1

fullbuffer " = 0

* semaphore এর অবশ্যই atomic হবে।

Sem-2

* Shared buffer with N elements

emptybuffer initialize = N

fullbuffer " = 0

N = 5

	E(5)	F(0)
P()	4	1
P()	3	2
C()	4	1
C()	5	0
C()	5	-1

Sem-3

* Multiple producer, consumer
* Shared buffer with N elements

Requirement:

① Each consumer must grab unique filled element

② " producer " " " empty "

* my-i , my-j

Sem-3

25.01.23

* Multiple producer, consumer
* Shared buffer with N elements

Requirement:

① Each consumer must grab unique filled element

② " producer " " " empty "

* my-i , my-j shared value.

*producer/consumer-এর same lock.

Consumer #1

```
sem_wait(&mutex);
sem_wait(&fullbuffer);
my_j = findFull(&buffer);
```

→ এখানে context switch হলে consumer lock সহ ঘুমায়ে যাবে। producer বারবার এসে ঘুরে lock-acquired. ⇒ Deadlock

Deadlock vs livelock Important for exam

Deadlock out of syllabus.

* **Livelock:** Both trying to access same shared resource but failing. (Dining philosopher)

* **Deadlock:** One waiting for another to finish using the resource.

* producer/consumer আলাদা lock নিলেও race হতে পারে।

* প্রত্যেক my-i, my-j access করার জন্য lock নিব।

READER/WRITER LOCK:

* multiple producer can work parallely in producer/consumer problm. Here, only one writer can write at a time.

* multiple consumer cannot work parallely in p/c problm. Here, multiple reader can read at a time.

* What if we need to prioritize reader/writer?