IT5002 Tutorial 8

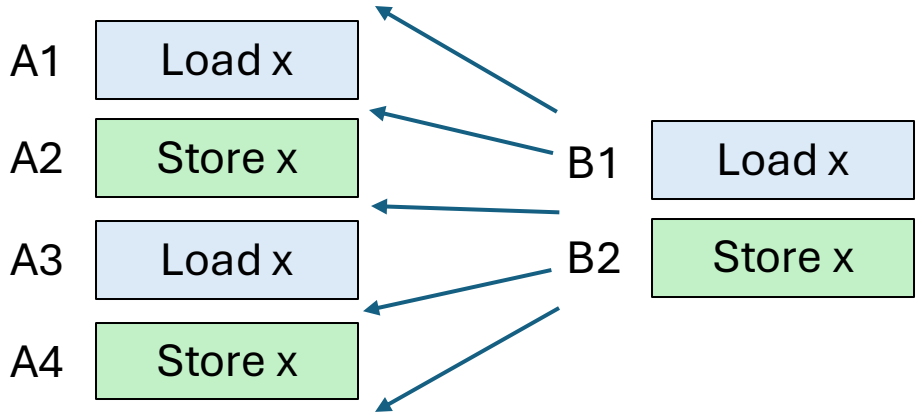
AY 2025/26 Semester 1

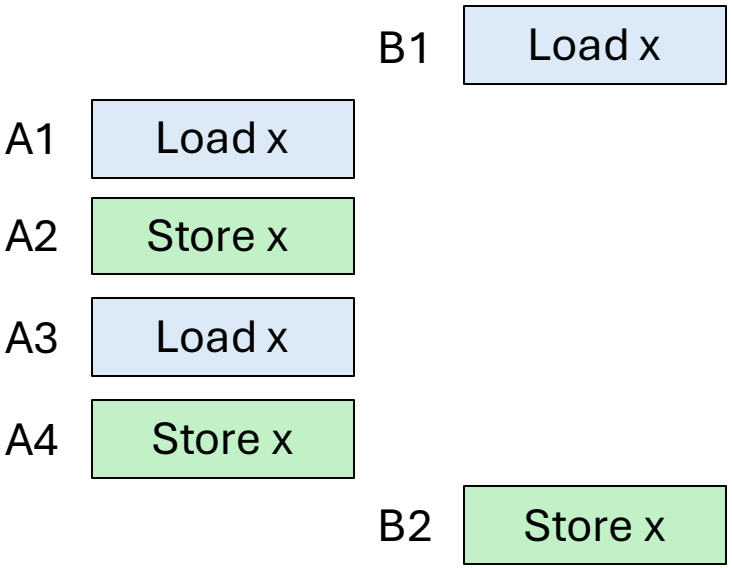
Prepared by Michael Yang

Slides adapted from Theodore Leebrant, Prof. Colin and Prof. Aaron

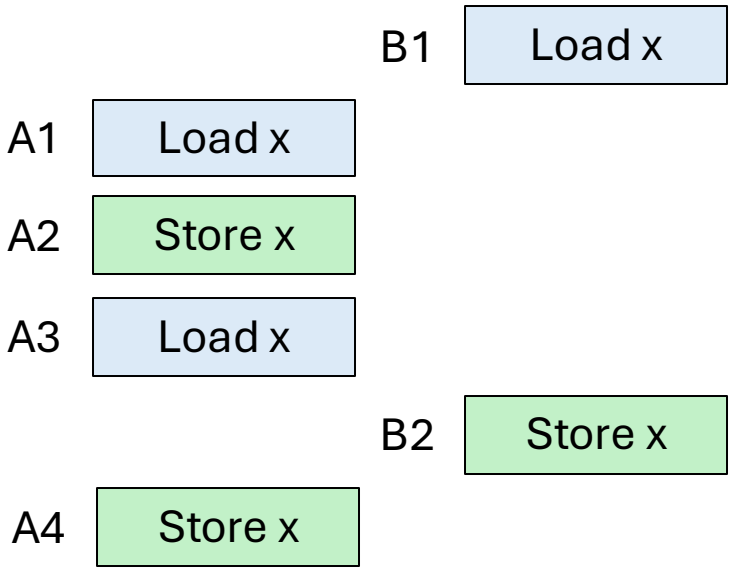
1. Two tasks A and B are run concurrently using a shared variable x. Assume that load and store of x is atomic, x is initialized to 0, and x must be loaded into a register before further computation can take place. How many relevant interleaving scenarios are there? What are all possible values of x after both tasks have terminated?

Within task A and B, the instructions cannot be rearranged, so consider how many unique ways are there to interleave the instructions B1 and B2 with task A





Final value: x = 0



Final value: x = 2

Listing all possible outcomes,

B1-A1-A2-B2-A3-A4 x = 1
B1-A1- B2-A2-A3-A4 x = 2
B1-B2-A1-A2-A3-A4 x = 2
A1-B1-A2-A3-A4-B2 x = 0
A1-B1-A2-A3-B2-A4 x = 2
A1-B1-A2-B2-A3-A4 x = 1
A1-B1-B2-A2-A3-A4 x = 2
A1-A2-B1-A3-A4-B2 x = 2
A1-A2-B1-A3-B2-A4 x = 2
A1-A2-B1-B2-A3-A4 x = 3
A1-A2-A3-B1-A4-B2 x = 2
A1-A2-A3-B1-B2-A4 x = 2
A1-A2-A3-A4-B1-B2 x = 4

So the possible values are 0, 1, 2, 3, and 4.

2. Three tasks are executing using semaphores S1 and S2 and a shared variable x. Assume S1 is initialized to 1, S2 is initialized to 0, and x is initialized to 0. What are the possible values of x after all three tasks have terminated?

A	B	C
P(S2); P(S1); x = x*2; V(S1);	P(S1); x = x * x; V(S1);	P(S1); x = x + 3; V(S2); V(S1);

Note that S2 is initialized to 0. Task A can only proceed when another tasks signals S2, and task C is the only one that can signal S2. Furthermore, task A can only proceed after task C signals S1.

So task A will always execute after task C.

So the possible permutations are:

B-C-A: x = 6
C-A-B: x = 36
C-B-A: x = 18

3. Use semaphores to implement a one-time use Barrier() without using any form of loops.

```
int arrived = 0;      //shared variable
Semaphore mutex = 1; //binary semaphore to provide mutual exclusion
Semaphore waitQ = 0; //for N-1 process to blocks

Barrier(N) {
    wait(mutex);
    arrived++;
    signal(mutex);
    if (arrived == N) signal(waitQ);
    wait(waitQ);
    signal(waitQ);
}
```

4. A platform X does not support semaphores or mutexes, but it supports the atomic function `bool _sync_bool_compare_and_swap(int* t, int v, int n)`. Use it to implement `atomic_increment`. Your function should always return the incremented value of referenced location t, and to be free of race conditions. The use of busy waiting is allowed.

Compare and swap: if t is equal to temp, set t = temp + 1 and return true. Else return false.

```
int atomic_increment(int* t) {
    do {
        int temp = *t;
    } while (!_sync_bool_compare_and_swap(t, temp, temp+1));
    return temp+1;
}
```

If the CAS fails (perhaps updated by another thread), in the next iteration of the loop, temp is updated to the 'latest' value of t. So the loop keeps trying to update t until it knows it has succeeded.

5. Consider the following segment table. What are the physical addresses for the following logical addresses? Which of these addresses, if accessed, would result in a segmentation violation?

Segment	Base	Length
0	219	600
1	2300	14
2	1327	580
3	1952	96

(a) (0,430)
219+430, legal access

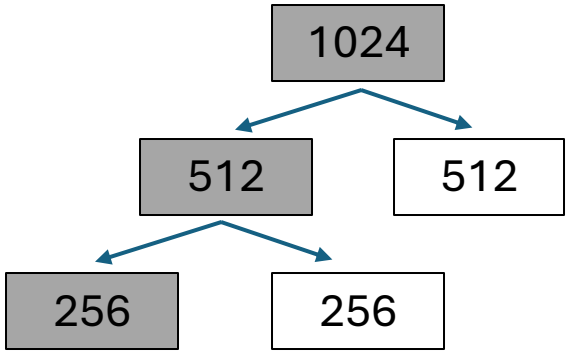
(b) (1,10)
2300+10, legal access

(c) (2,500)
1327+500, legal access

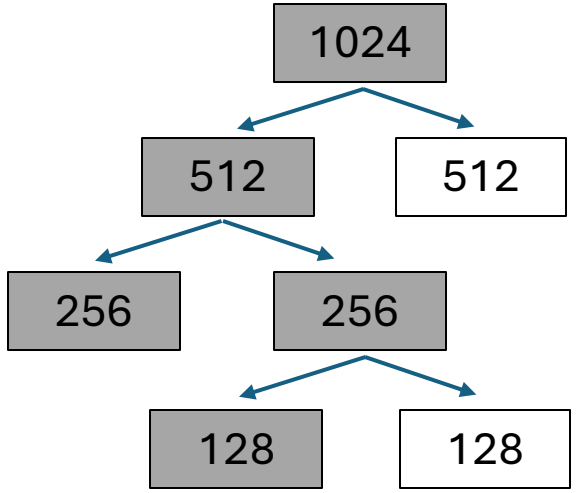
(d) (3,400)
1952+400, illegal access

6. Assume there is a 1,024KB segment where memory is allocated using the buddy system. Describe the configuration of the system as the following allocation requests are served.

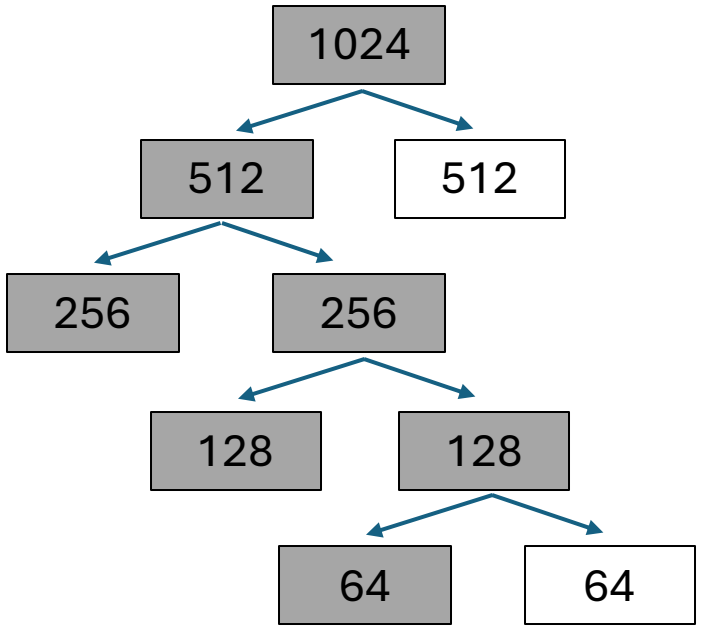
(a) Request 240 kB



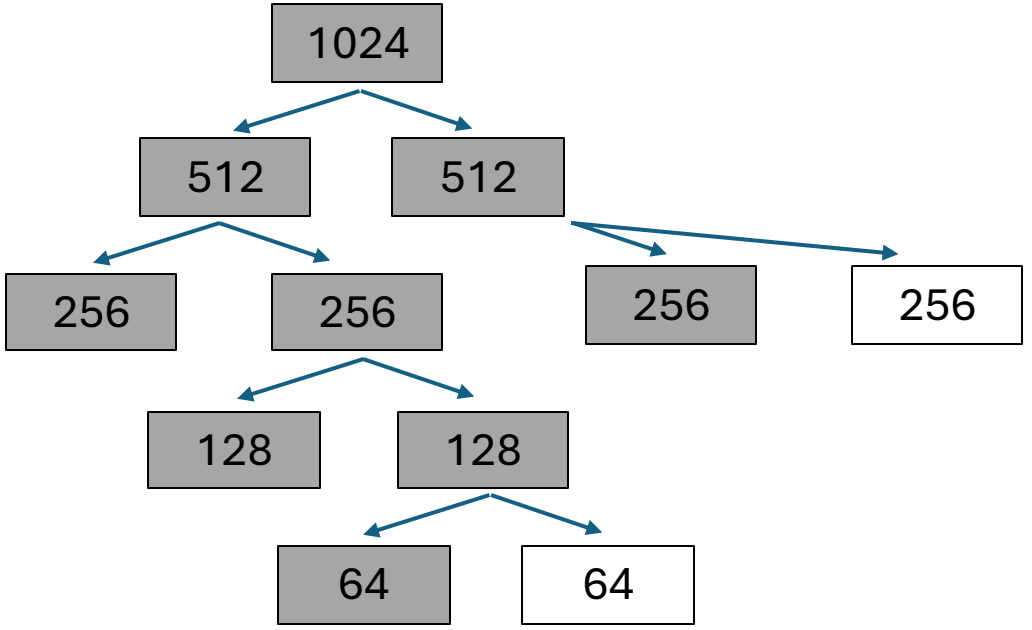
(b) Request 120 kB



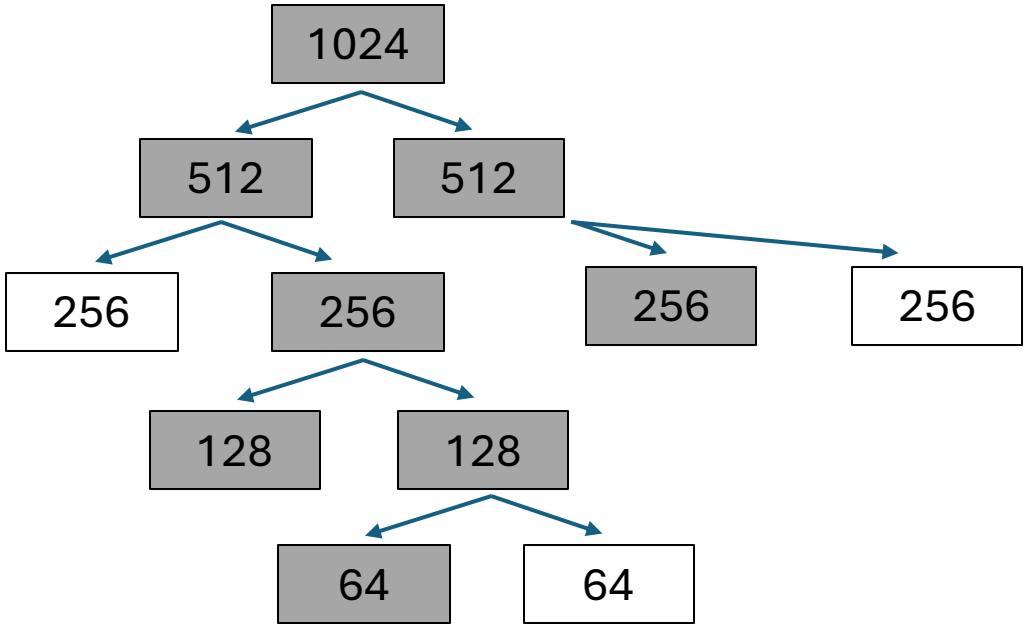
(c) Request 60 kB



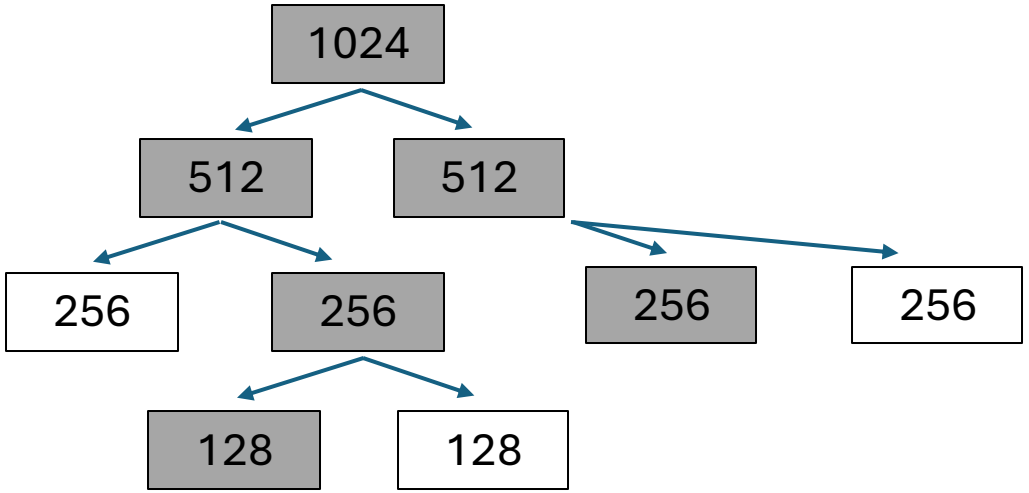
(d) Request 130 kB



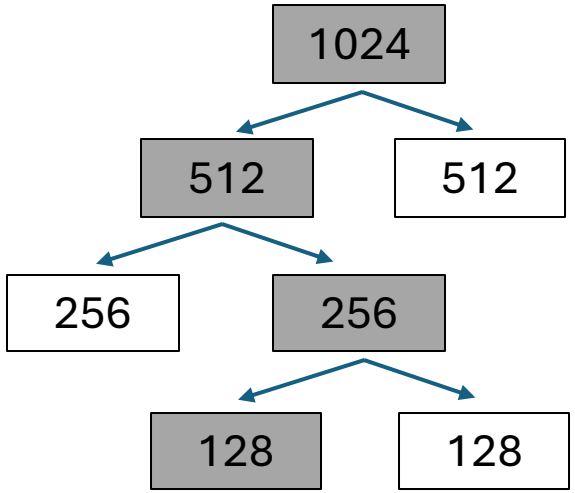
Free (a)


1024
512
512
256
256
256
256
128
128
64
64

Free (c)


1024
512
512
256
256
256
256
128
128

Free (d)



Slides uploaded to https://github.com/michaelyql/IT5002

Email: e1121035@u.nus.edu

Anonymous feedback: https://bit.ly/feedback-michael

(or scan the QR below)