HW4

1. Ans
2. Since each processor just lock and unlock once as it is test-and-test&set for which processor don’t need to do test&set every time released. There are only one transaction.
3. 100 cycles; 1600 cycles
4. It’s better to give priority to the process according to the order, in which way we can ensure one transaction once and saving bus traffic.

(e) No. Because cache coherence is no more needed.

1. Ans
2. (i) F&I

ticket lock:

init ticket=now\_s=0;

lock:

iTicket = F&I (&(ticket));

while(iTicket != now\_s);

unlock:

now\_s++;

array-based lock:

init location=0;

lock:

iLocation = F&I (&(location));

while(array[iLocation] == lock);

array[iLocation] = lock;

unlock:

array[iLocation+1] = unlock;

(ii)LL-SC

Ticket lock:

ll reg1 &ticket

add ticket ticket 1

lock: ll reg2 &now\_s

bnz reg1 reg2 lock

ret

unlock: add now\_s now\_s 1

ret

array-based lock:

lock: ll reg1 array[location]

add location location 1

bnz reg1 lock

sc array[reg1] #1

ret

unlock: st array[reg1+1] #0

ret

1. Yes, the trick is for an arriving process, fetch & store last processor node and check if there is predecessor. If so, wait until it is unlocked. And for an unlock, if there is successor, set it “unlock”.

1. Ans

ll reg1 location

bne reg1 reg2 swap

ret

swap: sc location reg2

ret

1. The code

void barrier()

{

pthread\_mutex\_lock (&mutex);

waiting--;

if (waiting > 0)

pthread\_cond\_wait (&conditon, &mutex);

else {

waiting = NumThread;

pthread\_cond\_broadcast (&condition);

}

pthread\_mutex\_unlock (&mutex);

}