Name: Bingying Liang

Student ID: 48999397

course Number: CS7343

Distance student:

1. Sol: ① available

u	v	w	x
3	4	2	0

NEED

A	0	0	0	1
B	1	1	2	2
C	0	4	5	1
D	5	0	0	0
E	3	3	0	0

E can satisfy. so E.

after E finish, $(3,4,0,0) - (3,3,0,0) + (4,3,2,1) = (0,1,0,0) + (4,3,2,1)$

$= (4,4,2,1)$

available

u	v	w	x
4	4	2	1

② NEED

A	0	0	0	1
B	1	1	2	2
C	0	4	5	1
D	5	0	0	0
E	0	0	0	0

A can satify so A

after A finish, $(4,4,2,1) - (0,0,0,1) + (1,1,0,3) = (4,4,2,0) + (1,1,0,3)$

$= (5,4,2,3)$

available

u	v	w	x
5	4	2	3

③ NEED

A	0	0	0	0
B	1	1	2	2
C	0	4	5	1
D	5	0	0	0
E	0	0	0	0

Bing ying Liang
ID: 48959397

Now B. D can satisfy, so B go first.

after B finish, $(5,4,2,3)-(1,1,2,2)+(1,2,4,5)=(4,3,0,1)+(1,2,4,5)$
$=(5,5,4,6)$

available

u	v	w	x
5	5	4	6

④ NEED

A	0	0	0	0
B	0	0	0	0
C	0	4	5	1
D	5	0	0	0
E	0	0	0	0

D can be satisfy so D.

after D finish $(5,5,4,6)-(5,0,0,0)+(5,0,1,1)=(0,5,4,6)+(5,0,1,1)$
$=(5,5,5,7)$

available

u	v	w	x
5	5	5	7

④ NEED

A	0	0	0	0
B	0	0	0	0
C	0	4	5	1
D	0	0	0	0
E	0	0	0	0

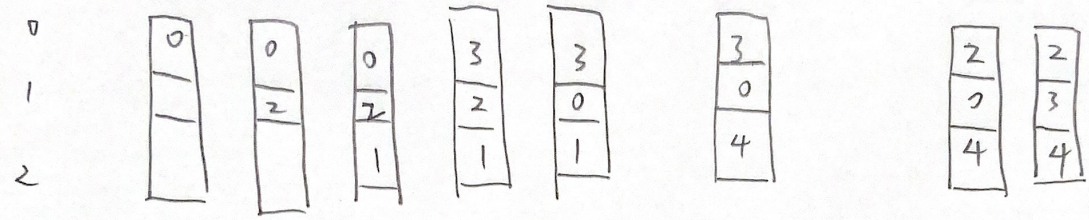
C can be satisfied. C can finish.

Therefore, it is a safte state. it can be order by E, A, B, D, C.

2. sol: 0 2 1 3 0 1 4 0 1 2 3 4

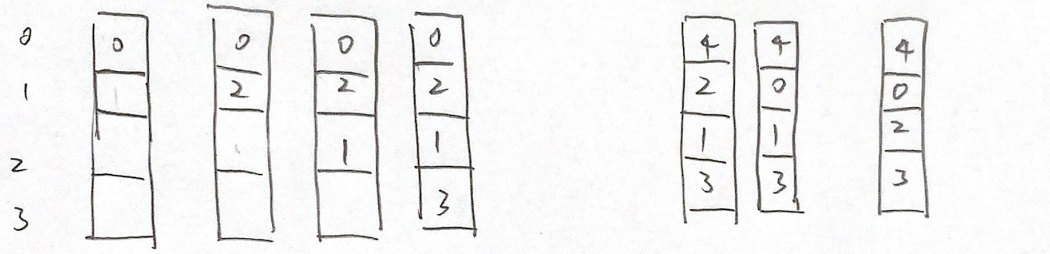
a)

scheme1:



∴ 8 pages faults

scheme2:



∴ 7 pages faults.

b) The pages faults decreases with larger pages.

3. Sol: to deadlock to occur necessary conditions:

① Mutual Exclusion

② Hold and wait

③ No preemption

④ Circular Wait

① Mutual Exclusion that cannot be prevented by the OS. Because it must supported by the OS. In general, this condition is hard to break. If access to resource requires mutual exclusion, then mutual exclusion must be supported by the OS; Some resources, such as files, may allow multiple access for read but only exclusive access for writes.

We can break ②, ③, ④ in the following ways.

② Hold and wait to break: request a process all of its required resources at one time.

③ No preemption to break: Process must release resource and request again; OS may preempt a process to require it releases its resources.

④ Circular wait to break: Define a linear ordering of resource types.

Bingying Liang
48991397

16 bit 4k pages = 4 x 1024 bytes.

4. Sol:

process 1

0	3 = 11
1	7
(2)	1
3	5

process 2

0	2 = 10
1	0
2	6
(3)	4

page table.

$2^6 = 64$ pages

4 x 1024

$\frac{2^6}{2^2} = 2^{(4)}$

Process	Address	Page #	offset	Physical address
Process 1	11,034	0010 (2)	1011000010	0001 1011000010
Process 2	12,345	0011 (3)	0000011111001	0100 0000 0011 1001

11,034 = 0010 1011 0001 0010

4 bit

12,345 = 0011 0000 0011 1001

4 bit

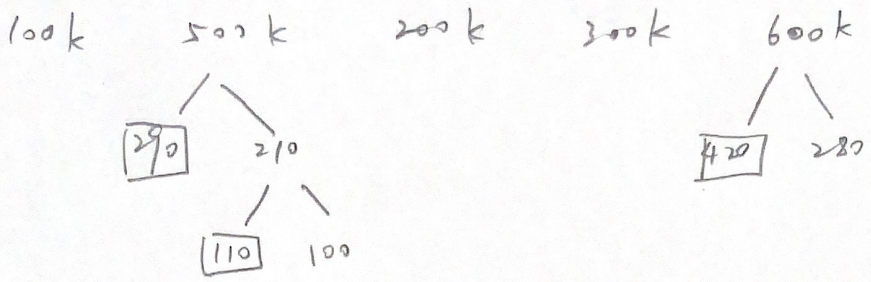
Biyying Liang

48993917

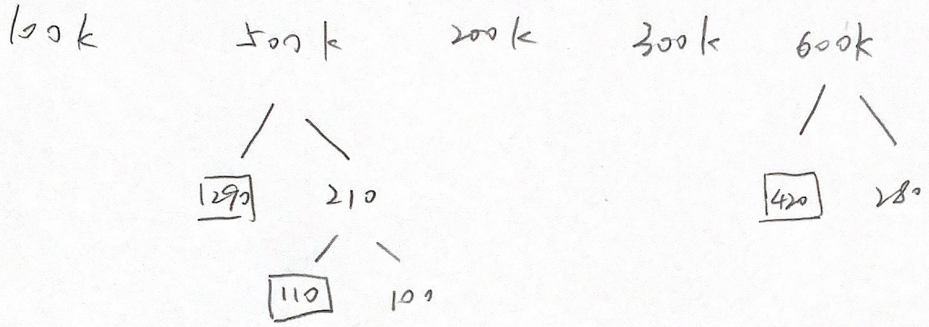
allocate 290, 420, 110, 330.

5. Sol:

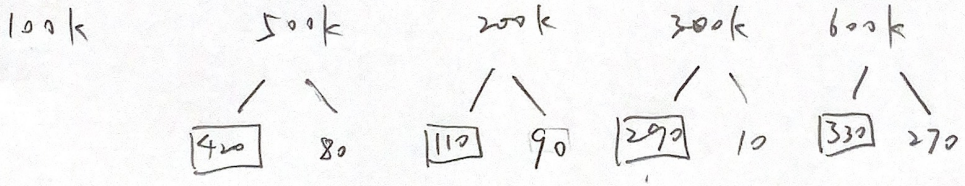
first fit:



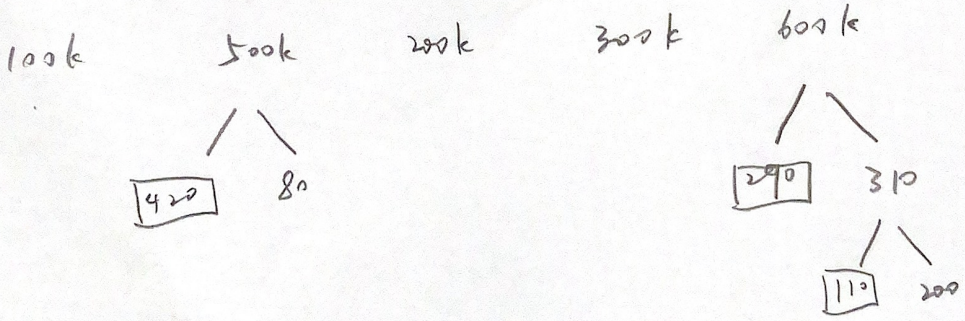
next-fit:



best fit



worst fit



Binyin Liu
48999397

6. sol: memory : 64 KiB OS + data : 16 KiB

P_1: 19 KiB code, 5 KiB data

P_2: 15 KiB code, 8 KiB data.

(a) Pages: No.

$64 \div 2 = 32$ pages

$16 \div 2 = 8$ pages

Available pages for processes: $32 - 8 = 24$ pages.

P_1: $\lceil \frac{19}{2} \rceil = 10$ pages $\lceil \frac{5}{2} \rceil = 3$ pages P_1 needs: 13 pages

P_2: $\lceil \frac{15}{2} \rceil = 8$ pages $\lceil \frac{8}{2} \rceil = 4$ pages P_2 needs: 12 pages

$13 + 12 = 25 > 24$ pages.

∴ These processes can't fit in physical memory if 2 KiB pages are used.

(b) Segmentation: Yes

$64 - 16 = 48$ KiB available for processes.

Two processes needs: $19 + 5 + 15 + 8 = 47 < 48$

∴ These processes fit in physical memory if pure segmentation is used.

Bing-ying Liang
48999397

7. Semaphore n=0; s=1

```
void producer(){
}
                while (true)
                {
                 producer();
                 semWait(s);
                 append();
                 semSignal(s);
                 semSignal(n);
                }
}

void consumer(){
{
                while (true)
                {
                 semWait(n);
                 semWait(s);
                 take();
                 semSignal(s);
                 consume();
                }
}

void main()
{
      parbegin(producer, consumer);
}
```

Bingying Liang
48599397

8.

A	B	C	D	
f[1]	f[0]	Turn	State	
0	0	0	because one of flags has to be true.	: impossible
0	0	1	impossible, because one of flags have to be true.	: Impossible
0	1	0	Impossible, because. p1: flag[1]=0 means p1 has already executed its critical section, which means the turn must be 1.	: Impossible
0	1	1	possible: p1 is done, p0 is waiting p0: flag[0]=1 means p0 go next, for p1	: p0 waiting, p1 done
1	0	0	possible: p0 done, p1 is waiting. p1: flag[1]=1 means p1 wants to go, turn could be 0. because it's going to set it to basically 0).	: p0 done, p1 waiting
1	0	1	Impossible: if flag[1]=1, turn should be 0.	: Impossible
1	1	0	possible: p0 is in, p1 is waiting	: p0 and p1 waiting
1	1	1	possible: p1 is in p0 is waiting	: p1 and p0 waiting.

Binjing Liang
48999397

9. sol: ① if the question doesn't has any wrong.

the answer that it should be always be 0.

```
int sum = 0
for( count = 1 ; count <= n; count ++){
   sum = tally + sum;
}
```

sum = 0 + 0 = 0.

which means tally never change.

lw
add
sw

② if the question means tally ++ which means tally +1 every time:

start with: tally = 0

the first parbegin (total(), total()) lower bound = 2, upper bound = 100.

start with: tally = 2 or 100

the second parbegin (total(), total())

① if tally = 2 lowerbound: 4 upperbound = 104

② if tally = 100 lowerbound: 102 upperbound: 200

∴ therefore, the second parbegin lower bound = 4 upperbound = 200.

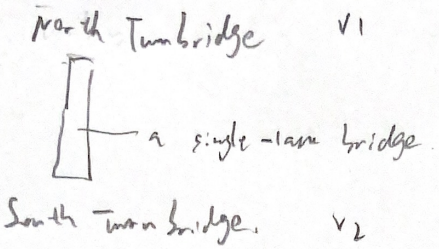
the third parbegin (total() total())

① if tally: 4 lower bound = 6 upperbound = 106

if tally = 200 lower bound = 202 upperbound: 300

∴ Therefore the final value of tally lower bound = 6, upperbound = 300.

10. sol: (a)



Bingyi Liu
48999397

```
sem_t m;
sem_init (&m, 0, 1); //initialize semaphore.

sem_wait (&m);
//critical section
using the bridge
sem_post (&m);
```

m	V1	State	V2	State
1		running		ready
1	use sem_wait()	running		ready
0	sem_wait() return	running		ready
0	using bridge	running		ready
0	switch to V2	ready		running
0		ready	use sem_wait()	running
		ready	sem -1	running
-1		ready	(sem<0) → sleep	sleeping
-1		running	switch to T0	sleeping
-1				sleeping
-1	finish using bridge	running		sleeping
-1	sem_post()	running		sleeping
0	sem +1	running		ready
0	tell the V2	running		ready
0	sem_post return	running		running
0	switch to T1	ready		
0		ready	sem_wait() return	running
0		ready	use bridge	running

0	ready	use sem_post()	running
0	ready	sem_post return	running

b). Starvation-free, we can add buffer to like anyarray to schedule the order of farmers.

```
int buffer[MAX];
```

and use the first in first out to let farmers to use the bridge..

Bingqing Liang
48599357