1. FILL IN BLANKS (10 points)

(1) interrupt (2) trap (3) system calls (4) privilege

(5) process (6) waiting . (7) kernel / system / supervisor / privileged

(8) message passing (9) signal() (10) bounded waiting

2. CHOICE (10 points) C C C A D D D C C B

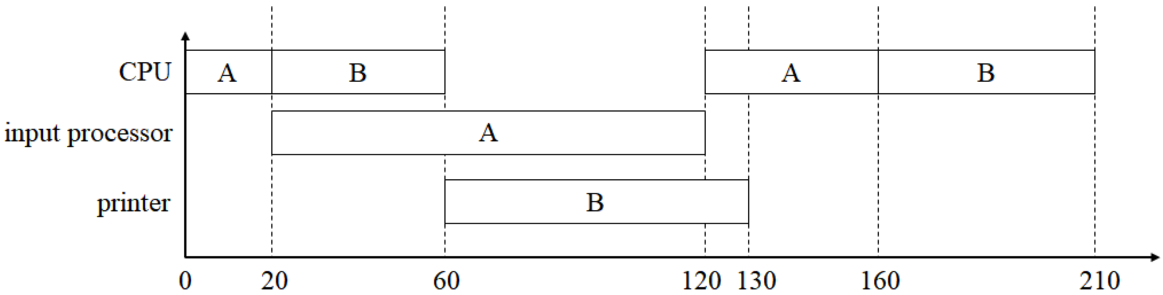
3. ESSAY QUESTIONS (20 points)

(1) (6 points) The Bounded-Buffer producer-consumer problem
The Readers-Writers Problem
The Dining-Philosophers Problem

(2) （6 points） (a) initial value: 1, maximum value: 1, minimum value: -9
(b) initial value: 3, maximum value: 3, minimum value: -7

(3) （8 points）

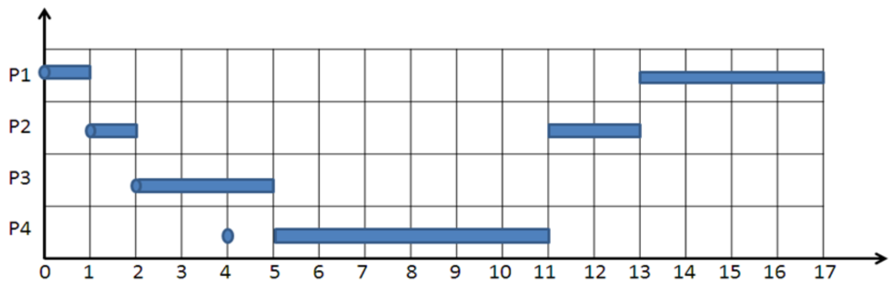
(1)

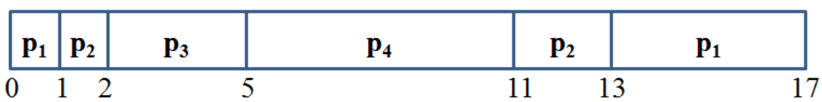


(2) for process A: waiting time: 0ms turnaround time: 160 ms
for process B: waiting time: 50ms turnaround time: 210 ms

4. (20 points) (1) *preemptive* priority scheduling algorithms.

(a)



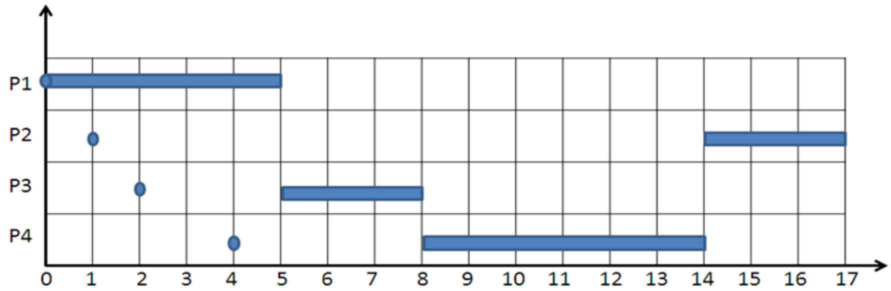


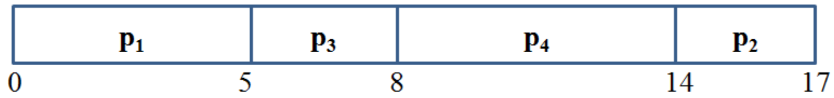
(b) the average waiting time: (12+9+0+1)/4=5.5

(c) the average turnaround time: (17+12+3+7)/4=9.75

(2) ***non-preemptive*** **priority scheduling algorithms:**

(a)





(b) the average waiting time: (0+13+3+4)/4=5

(c) the average turnaround time: (5+16+6+10)/4=9.25

5. (20 points)

SEMAPHORES

Splate=1; Used for mutual exclusion use of the plate.

Sapple=0; Used for synchronization between the process father and son.

Sorange=0; Used for synchronization between the process mother and daughter.

MUTEX=1; Used for mutual exclusion operation on variable orange_count.

Orange_ROOM=3; Used to record the rooms left for keeping oranges.

VARIABLE

orange_count=0; Used to record the number of oranges kept in the plate.

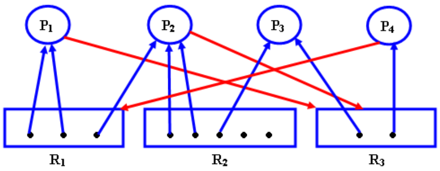
CODE SECTIONS:

Father: **while(true){** **(1) wait(Splate);** **Put an apple into the plate;** **(2) signal(Sapple);** **}**	**Son:** **while(true){** **(3) wait(Sapple);** **Take the apple from the plate;** **(4) signal(Splate);** **eats the apple;** **}**
Mother: **while(true){** **5) wait(orange_ROOM);** **wait(MUTEX);** **orange_count++;** **if (orange_count==1) wait(Splate);** **signal(MUTEX);** **Put an orange into the plate;** **(6) signal(Sorange);** **}**	**Daughter:** **while(true){** **(7) wait(Sorange);** **Take an orange from the plate;** **(8) wait(MUTEX);** **orange_count--;** **if (orange_count==0) signal(Splate);** **signal(MUTEX);** **signal(orange_ROOM);** **eats the orange;** **}**

6. (20 points)

(1) Total resources in the system are (R1, R2, R3) = (3, 5, 2),

(2)



(3) System is in an unsafe state

Need=

	needs		
	R_1	R_2	R_3
P_1	0	0	1
P_2	1	3	2
P_3	1	3	1
P_4	2	0	0

Resources available=(0, 2, 0) can not satisfy the requirements need from any processes, so there exists no safe sequence of processes, so system is in an unsafe state.

(4)

System is not deadlocked.

According to deadlock detection algorithm,

Work=(0,2,0) finish[i]=false for i=1,2,3,4

Because request3=(0,0,0)<need3 and request3<work

So, finish[3]=true, work=(0,3,1)

Because request1=(0,0,1)<=need3 and request1<work

So, finish[1]=true, work=(2,3,1)

Because request2=(0,0,1)<need2 and request2<work

So, finish[2]=true, work=(3,5,1)

Because request4=(1,0,0)<need4 and request4<work

So, finish[4]=true, work=(3,5,2)

Finish[i]=true for all i, so the system is not deadlock.