Section A (32 marks)

Attempt ALL questions from this Section. In case of the fill-in-the-blank questions, abbreviation or short form will NOT be accepted.

1. [2 marks]
At the beginning of each instruction cycle, the processor fetches the instruction whose address is stored in (i)_____ and the fetched instruction is loaded into (ii)_____.
a) accumulator
b) arithmetic and logic unit
c) instruction register
d) program counter

Ans: d and c

2a. [1 mark]
_____ enables devices and controllers to transfer blocks of data to and from main memory directly.
a) Programmed I/O
b) Direct memory access (DMA)
c) Interrupt-driven I/O
d) None of the above

Ans: b

2b. [1 mark]
In _____, a process has to busy wait for the I/O operation to be completed before proceeding.
a) Programmed I/O
b) Direct memory access (DMA)
c) Interrupt-driven I/O
d) None of the above

Ans: a

3a. [3 marks]
Complete the following table by inserting "high", "middle" or "low".

Type of memory	Cost per bit	Access time
Disk	(i)	(iv)
Main memory	(ii)	(v)
Registers	(iii)	(vi)

Type of memory	Cost per bit	Access time
Disk	**Low**	**High**
Main memory	**Middle**	**Middle**
Registers	**High**	**Low**

3b. [3 marks]
Complete the following table by inserting “high”, “middle” or “low”.

Type of memory	Capacity	Access frequency by processor
Disk	(i)	(iv)
Main memory	(ii)	(v)
Registers	(iii)	(vi)

Type of memory	Capacity	Access frequency by processor
Disk	**High**	**Low**
Main memory	**Middle**	**Middle**
Registers	**Low**	**High**

4a. [2 marks]
After the I/O device issues an interrupt signal to the processor, the following events will happen. Write down the correct sequence of those events.

(i) Processor loads new PC value based on interrupt
(ii) Processor pushes PSW and PC onto control stack
(iii) Processor restores PSW and PC from the control stack
(iv) Process the interrupt
(v) Processor finishes execution of current instruction

(v), (ii), (i), (iv), (iii)

4b. [2 marks]
After the I/O device issues an interrupt signal to the processor, the following events will happen. Write down the correct sequence of those events.

(i) Processor loads new PC value based on interrupt
(ii) Processor restores PSW and PC from control stack
(iii) Processor pushes PSW and PC onto the control stack
(iv) Process the interrupt
(v) Processor finishes execution of current instruction

(v), (iii), (i), (iv), (ii)

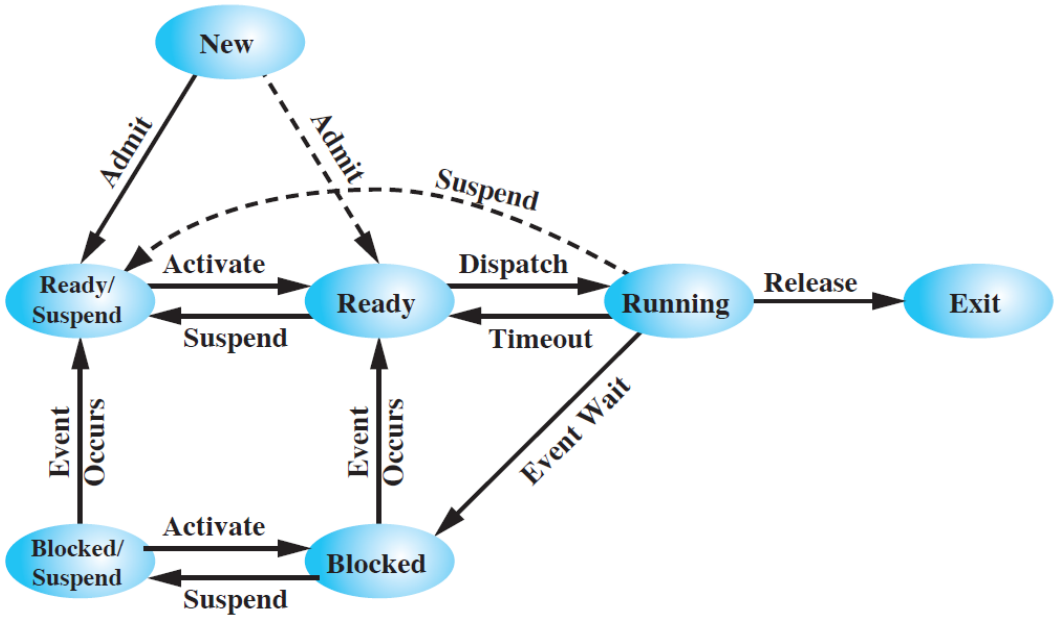
5a. [1 mark]
Comparing with uniprogramming, multiprogramming generally achieves _____ throughput and _____ mean response time.
a) higher, higher
b) higher, lower
c) lower, higher
d) lower, lower

Ans: b

5b. [1 mark]
Comparing with uniprogramming, multiprogramming generally achieves _____ mean response time and _____ throughput.
a) higher, higher
b) higher, lower
c) lower, higher
d) lower, lower

Ans: c

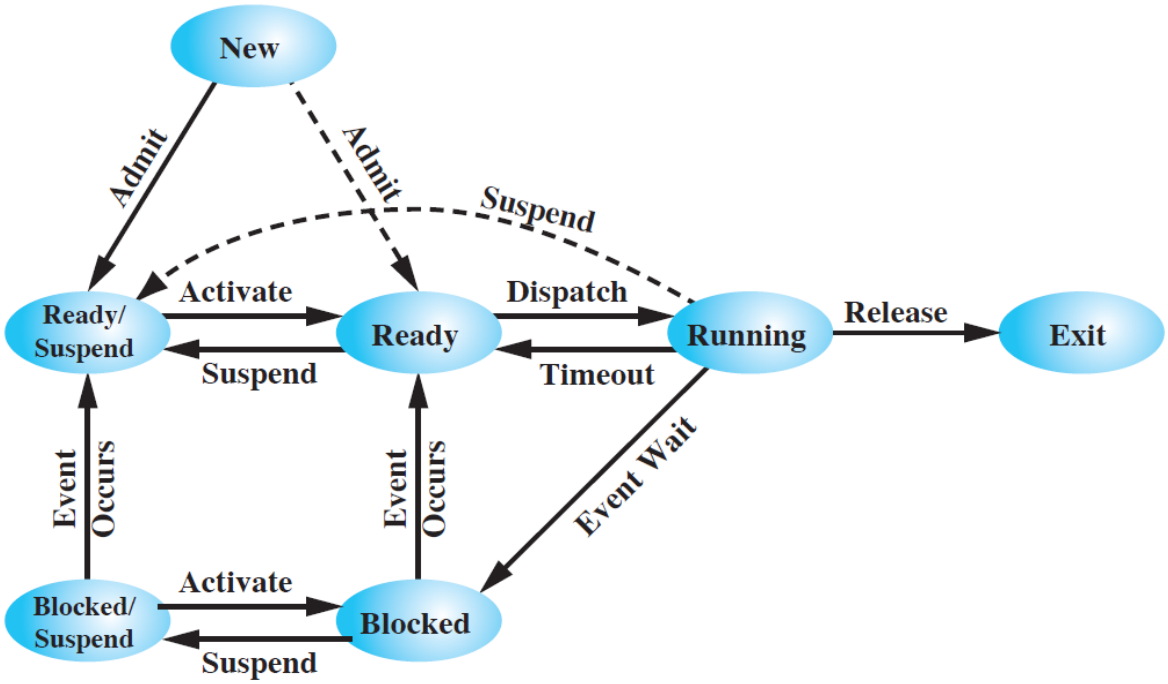
6a. [3 marks]
Consider the following 7-state process model. Complete the table by filling the state transition (may or may not be shown on the diagram). The first one is an example.



Event	Transition
The dispatcher chooses a process to run.	Ready → Running
An I/O operation for which a process has been waiting completes.	(i)
A process invokes an I/O operation that must be completed before it can continue.	(ii)
OS swaps out a process waiting for an event to occur.	(iii)

Event	Transition
The dispatcher chooses a process to run.	Ready → Running
An I/O operation for which a process in memory has been waiting completes.	**Blocked → Ready**
A process invokes an I/O operation that must be completed before it can continue.	**Running → Blocked**
OS swaps out a process waiting for an event to occur.	**Blocked→ Blocked/Suspend**

6b. [3 marks]
Consider the following 7-state process model. Complete the table by filling the state transition (may or may not be shown on the diagram). The first one is an example.



Event	Transition
The dispatcher chooses a process to run.	Ready → Running
A process has reached its time slice.	(i)
OS brings in a high-priority ready process from disk into memory.	(ii)
A parent process terminates its child process waiting for an event to occur.	(iii)

Event	Transition
The dispatcher chooses a process to run.	Ready → Running
A process has reached its time slice.	**Running→Ready**
OS brings in a high-priority ready process from disk into memory.	**Ready/suspend → Ready**
A parent process terminates its child process waiting for an event to occur in memory.	**Blocked → Exit**

7. [1 mark]
The collection of process identification, processor state information and process control information is referred to as the _____ .

Ans: process control block

8a. [1 mark]
When a child process is spawned, the `fork` system call returns _____ to the parent process and _____ to the child process.
a) a value of zero, the PID of the child process
b) a value of one, the PID of the parent process
c) the PID of the child process, a value of zero
d) the PID of the child process, a value of one

Ans: c

8b. [1 mark]
When a child process is spawned, the `fork` system call returns _____ to the child process and _____ to the parent process.
a) a value of zero, the PID of the child process
b) a value of one, the PID of the parent process
c) the PID of the child process, a value of zero
d) the PID of the child process, a value of one

Ans: a

9. [1 mark]
Which of the following events would **not** lead to a ***trap*** being generated?
a) divided by zero
b) completion of a disk I/O
c) illegal file access attempt
d) reference a protected memory location

Ans: b

10. [1 mark]
Multithreading is important because it can _____.
a) facilitate software design and promote good programming practices
b) improve performance and scalability
c) facilitate cooperation/synchronization of activities
d) all of the above

Ans: d

11a. [1 mark]
Which of the following statement about multithreading is true?
a) A multithreaded solution using multiple user-level threads cannot achieve better performance on a multiprocessor system than on a single-processor system.
b) A multithreaded solution using multiple kernel-level threads cannot provide better performance than a single-threaded solution on a single-processor system.
c) Kernel-level threads are faster than user-level threads in performing thread switching within a process.
d) When a user-level thread is blocked, not all of the threads within the process are blocked.

Ans: a

11b. [1 mark]
Which of the following statements are correct description of kernel-level threads (KLTs)?

(i) The kernel is aware of the existence of KLTs
(ii) KLTs are created by invoking an application-level function.
(iii) The transfer of control from one thread to another within the same process requires a mode switch to the kernel is one of the advantages of the KLT approach.
(iv) Using KLTs cannot provide a better performance than a single-threaded solution on a single-processor system.

a) (i), (iii) and (iv)
b) (i) only
c) (ii) only
d) (i) and (iv)

Ans: b

12a. [2 marks]
Complete the following table by indicating whether the state is for process only, thread only, or both.

State	process/thread/both
Suspend	(i)
Blocked	(ii)

State	process/thread/both
Suspend	**Process**
Blocked	**Both**

12b. [2 marks]
Complete the following table by indicating whether the state is for process only, thread only, or both.

State	process/thread/both
Ready	(i)
Suspend	(ii)

State	process/thread/both
Ready	**Both**
Suspend	**Process**

13. [1 mark]
_____ restricts access to a shared variable to only one thread at any given time.

Ans: mutual exclusion

14a. [1 mark]
A semaphore that does not specify the order in which processes are removed from the queue is a _____ semaphore.
a) strong
b) binary
c) counting
d) weak

Ans: d

14b. [1 mark]
A semaphore whose definition includes the policy that the process that has been blocked the longest is released from the queue first is called a _____ semaphore.
a) strong
b) binary
c) counting
d) weak

Ans: a

15. [1 mark]
When a process is in the critical section, it must be running.
a) True
b) False

Ans: b

16. [1 mark]
Which of the following statements about critical sections is false?
a) Only one thread at a time can execute the instructions in its critical section for a particular resource.
b) If one thread is already in its critical section, another thread must wait for the executing thread to exit its critical section before continuing.
c) Once a thread has exited its critical section, a waiting thread may enter its critical section.
d) All threads must wait whenever any critical section is occupied.

Ans: d

17. [1 mark]
A binary semaphore can always be replaced by a mutex.
a) True
b) False

Ans: b

18a. [2 marks]
What happens when a thread calls `semWait(S)` before entering its critical section, where `S` is a binary semaphore with current value of 1?
a) The thread is allowed to enter its critical section and `S` is set to 0.
b) The thread is blocked and `S` remains no change.
c) The thread is blocked and `S` is set to 0.
d) The thread is allowed to enter its critical section and `S` remains no change.

Ans: a

18b. [2 marks]
What happens when a thread calls `semWait(S)` before entering its critical section, where `S` is a binary semaphore with current value of 0?
a) The thread is allowed to enter its critical section and `S` is set to 1.
b) The thread is blocked and `S` remains no change.
c) The thread is blocked and `S` is set to 1.
d) The thread is allowed to enter its critical section and `S` remains no change.

Ans: b

19a. [1 mark]
If a semaphore was initialized to be 3, its current value is -2. The number of processes blocked on the semaphore is _____

Ans: 2

19b. [1 mark]
If a semaphore was initialized to be 2, its current value is -3. The number of processes blocked on the semaphore is _____

Ans: 3

20. [1 mark]
A _____ send is an example of _____ communication that requires the sender to wait for receipt notification before continuing program execution.
a) blocking, synchronous
b) blocking, asynchronous
c) nonblocking, synchronous
d) nonblocking, asynchronous

Ans: a

21a. [1 mark]
a. In a resource-allocation graph, _____ represent processes and _____ represent resources.
a) circles, squares
b) arrows, circles
c) squares, circles
d) arrows, squares

Ans: a

21b. [1 mark]
In a resource allocation graph, an arrow points from a _____ to a _____ to indicate that the system has allocated a specific instance of resource to the process.
a) square, circle
b) circle, dot in a square
c) circle, square
d) dot in a square, circle

Ans: d

22. [1 mark]
If a resource allocation graph can be reduced by all its processes, then _____.
a) there is no deadlock
b) starvation occurs
c) the system is in an unsafe state
d) none of the above

Ans: a

23. [1 mark]
If we have a cycle in a resource allocation graph, then _____.
a) there is a deadlock
b) there is no deadlock
c) a deadlock might exist
d) none of the above

Ans: c

24. [1 mark]
Which of the following condition cannot be disallowed for non-sharable resources in order to prevent deadlocks?
a) circular wait
b) no preemption
c) hold and wait
d) mutual exclusion

Ans: d

Section B (48 marks)

Attempt ALL questions from this Section

Q1a. [10 marks]

Consider the following program fragment.

```
long prog_global = 1;

void *myThreadFun(void *in)
{
        long fun_local = 2;
        ++fun_local;
        ++prog_global;
        cout << "Thread: " << fun_local << " " << prog_global << endl;
        pthread_exit(NULL);
}

int main()
{       // irrelevant variable declarations are removed here
        long main_local = 3;
        pid = fork();
        if (pid != 0)
        {
                wait(NULL);
                pthread_create(&tid, NULL, myThreadFun, NULL);
                pthread_join(tid, NULL);
                ++main_local;
                ++prog_global;
                cout << "Process1: " << main_local << " " << prog_global << endl;
        }
        else {
                ++main_local;
                ++prog_global;
                cout << "Process2: " << main_local << " " << prog_global << endl;
        }
        pthread_exit(NULL);
}
```

(i) [6 marks]
What would be the output of the program? Show the order and values clearly.

```
Process2: 4 2
Thread: 3 2
Process1: 4 3
```

(ii) [4 marks]
Explain the value(s) of the global variable, prog_global, in the above output.

- **When fork is called, a copy of the process image of the parent (including the global variable prog_global with value 1) is made for the child who subsequently increments it to 2. So, the variable prog_global also with value 1 in the parent and the thread created by the parent (in the if statement) is not affected by the change made the child.**
- **In contrast, the parent and the thread created by the parent have access to the same variable prog_global. So, they can see the changes made by each other. First, the thread increments it from 1 to 2 and the parent increments it from 2 to 3.**

Q1b. [10 marks]

Consider the following program fragment.

```
long prog_global = 3;

void *myThreadFun(void *in)
{
        long fun_local = 2;
        ++fun_local;
        ++prog_global;
        cout << "Thread: " << fun_local << " " << prog_global << endl;
        pthread_exit(NULL);
}

int main()
{       // irrelevant variable declarations are removed here
        long main_local = 1;
        pid = fork();
        if (pid != 0) {
                wait(NULL);
                pthread_create(&tid, NULL, myThreadFun, NULL);
                pthread_join(tid, NULL);
                ++main_local;
                ++prog_global;
                cout << "Process1: " << main_local << " " << prog_global << endl;
        }
        else {
                ++main_local;
                ++prog_global;
                cout << "Process2: " << main_local << " " << prog_global << endl;
        }
        pthread_exit(NULL);
}
```

(i) [6 marks]
What would be the output of the program? Show the order and values clearly.

```
Process2: 2 4
Thread: 3 4
Process1: 2 5
```

(ii) [4 marks]
Explain the value(s) of the global variable, prog_global, in the above output.

- **When fork is called, a copy of the process image of the parent (including the global variable prog_global with value 3) is made for the child who subsequently increments it to 4. So, the variable prog_global also with value 3 in the parent and the thread created by the parent (in the if statement) is not affected by the change made the child.**
- **In contrast, the parent and the thread created by the parent have access to the same variable prog_global. So, they can see the changes made by each other. First, the thread increments it from 3 to 4 and the parent increments it from 4 to 5.**

Q2a. [7 marks]

Consider the following program running in a uniprocessor system and the two processes, `P1` and `P2`, are running concurrently.

Line no.	
	`int x;`
	`void P(int i)`
	`{`
1	`x = 10;`
2	`x = x - 1;`
3	`x = x + 1;`
4	`if (x != 10)`
5	`printf ("x is %d", x);`
	`}`
	`void main()`
	`{`
	`parbegin (P(1), P(2));`
	`}`

Use the above program to illustrate the quote: "In a race condition, the loser wins." by showing a sequence of interleaved statements such that "`x is 10`" is printed. For each statement, specify the process (`P1` or `P2`) that executes the statement, statement line number and value of `x`. The first one is given below.

Process	Line number	x
P1	1	10
P1	**2**	**9**
P1	**3**	**10**
P2	**1**	**10**
P2	**2**	**9**
P1	**4**	**9**
P2	**3**	**10**
P1	**5**	**10**

"x is 10" is printed.

Q2b. [7 marks]

Consider the following program running in a uniprocessor system and the two processes, `P1` and `P2`, are running concurrently.

Line no.	
	`int x;`
	`void P(int i)`
	`{`
1	`x = 10;`
2	`x = x + 1;`
3	`x = x - 1;`
4	`if (x != 10)`
5	`printf ("x is %d", x);`
	`}`
	`void main()`
	`{`
	`parbegin (P(1), P(2));`
	`}`

Use the above program to illustrate the quote: "In a race condition, the loser wins." by showing a sequence of interleaved statements such that "`x is 10`" is printed. For each statement, specify the process (`P1` or `P2`) that executes the statement, statement line number and value of `x`. The first one is given below.

Process	Line number	x
P1	1	10
P1	**2**	**11**
P1	**3**	**10**
P2	**1**	**10**
P2	**2**	**11**
P1	**4**	**11**
P2	**3**	**10**
P1	**5**	**10**

"x is 10" is printed.

Q3a. [11 marks]

Refer to the following solution to the bounded-buffer producer/consumer problem using semaphore. Assume that buffer size is **10**.

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1, n= 0, e= sizeofbuffer;
void producer()
{
     while (true) {
          produce();
          semWait(e);
          semWait(s);
          append();
          semSignal(s);
          semSignal(n);
     }
}
void consumer()
{
     while (true) {
          semWait(n);
          semWait(s);
          take();
          semSignal(s);
          semSignal(e);
          consume();
     }
}
void main()
{
     parbegin (producer, consumer);
}
```

Consider the following two cases, assuming no process is waiting for the time being. Explain how **synchronization** can be achieved by showing all changes of the values of the three semaphores until both processes are done with a data item.

(i) A producer wants to append a data item to a **full** buffer.

(ii) Following (i), a consumer wants to a take a data item from the buffer.

(i)

- **Since the buffer is full, the values of the semaphores are initially s = 1, n = 10, e = 0.**
- **When the producer is trying to append a data item, it changes e = 0 → -1 and blocks on e. This ensures that the producer cannot append data to a full buffer.**

(ii)

- **When the consumer wants to take a data item, it changes n = 10 → 9 and s = 1 → 0 and takes a data item.**
- **After taking a data item, it changes s = 0 → 1, e = -1 → 0 and unblocks the producer. This ensures that the producer can resume appending data to a non-full buffer.**
- **The producer changes s = 1 → 0 and appends a data item.**
- **After appending a data item, the producer changes s = 0 → 1 and n = 9 → 10.**

Q3b. [11 marks]

Refer to the following solution to the bounded-buffer producer/consumer problem using semaphore. Assume that buffer size is **10**.

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1, n= 0, e= sizeofbuffer;
void producer()
{
     while (true) {
          produce();
          semWait(e);
          semWait(s);
          append();
          semSignal(s);
          semSignal(n);
     }
}
void consumer()
{
     while (true) {
          semWait(n);
          semWait(s);
          take();
          semSignal(s);
          semSignal(e);
          consume();
     }
}
void main()
{
     parbegin (producer, consumer);
}
```

Consider the following two cases, assuming no process is waiting for the time being. Explain how **synchronization** can be achieved by showing all changes of the values of the three semaphores until both processes are done with a data item.

(i) A consumer wants to take a data item from an **empty** buffer.

(ii) Following (i), a producer wants to append a data item to the buffer.

(i)

- **Since the buffer is empty, the values of the semaphores are initially s = 1, n = 0, e = 10.**
- **When the consumer is trying to take a data item, it changes n = 0 → -1 and blocks on n. This ensures that the consumer cannot take data out of an empty buffer.**

(ii)

- **When the producer wants to append a data item, it changes e = 10 → 9 and s = 1 → 0 and appends a data item.**
- **After appending a data item, it changes s = 0 → 1, n = -1 → 0 and unblocks the consumer. This ensures that the consumer can resume taking data out of a non-empty buffer.**
- **The consumer changes s = 1 → 0 and takes a data item.**
- **After taking a data item, the consumer changes s = 0 → 1 and e = 9 → 10.**

Q4. [10 marks]

Consider the following solution using ***semaphores*** to the *one-writer many-readers* problem.

```
int         readcount;
Semaphore   sem_x, sem_y;
```

Writer

```
semWait(sem_x);
/* writing performed */
semSignal(sem_x);
```

Readers

```
semWait(sem_y);
readcount++;
if readcount==1 then semWait(sem_x);
semSignal(sem_y);
/* reading performed */
semWait(sem_y);
readcount--;
if readcount==0 then semSignal(sem_x);
semSignal(sem_y);
```

Suppose a *writer* is now writing in the system. No other readers are waiting for the time being.

(i) [2 marks]
What are the current values of the two semaphores?

- **sem_x = 0**
- **sem_y = 1**

(ii) [2 marks]
What will happen when a *reader* wants to read while the *writer* is still writing?

- **sem_y = 1 → 0**
- **readcount = 0 → 1**
- **sem_x = 0 → -1**
- **The reader blocks on sem_x**

(iii) [2 marks]
Following the *first reader*, what will happen when a *second reader* wants to read while the *writer* is still writing?

- **sem_y = 0 → -1**
- **The second reader blocks on sem_y**

(iv) [4 marks]
What will happen when the *writer* finishes writing?

- **sem_x = -1 → 0 by the writer, the first reader unblocks**
- **sem_y = -1 → 0 by the first reader, the second reader unblocks**
- **readcount = 1 → 2 and sem_y = 0 → 1 by the second reader**
- **both readers are reading**

Q5. [10 marks]

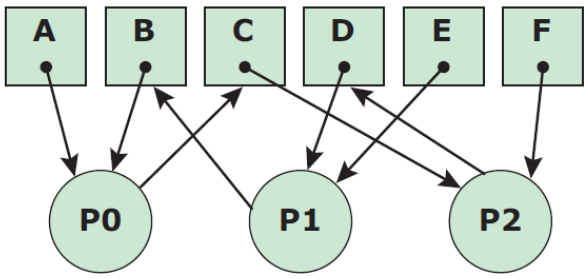
In the code below, three processes, `P0`, `P1`, and `P2` are competing for six resources `A` to `F`.

```
void P0()
{
  while (true) {
    get(A);
    get(B);
    get(C);
    // critical section:
    // use A, B, C
    release(A);
    release(B);
    release(C);
  }
}
```

```
void P1()
{
  while (true) {
    get(D);
    get(E);
    get(B);
    // critical section:
    // use D, E, B
    release(D);
    release(E);
    release(B);
  }
}
```

```
void P2()
{
  while (true) {
    get(C);
    get(F);
    get(D);
    // critical section:
    // use C, F, D
    release(C);
    release(F);
    release(D);
  }
}
```

(i) [6 marks]
Draw a ***resource allocation graph*** to show the possibility of a deadlock in this implementation. Show the cycle, if any, in the resource allocation graph.



The deadlock is illustrated by the cycle:
P0→C→P2→D→P1→B→P0.

(ii) [4 marks]
Which of the condition for deadlock can be prevented by modifying the order of some of the `get` requests? Show **one** possible modification.

Circular wait [1 mark]

One possible modification is as follows.

P0	P1	P2
A	B	C
B	D	D
C	E	F

- END -