1. Consider two functions, P and Q, and the main program creates two threads to execute P and Q as shown in the following code. Note that each statement is labeled and the label is just for convenience.

```plaintext
function P();
{ a = b + 1; print(a); }

function Q();
{ b = a + 3; print(b); }

main()
{ a = 5; b = 1;
  create a thread to execute P();
  create a thread to execute Q();
}
```

List all possible printouts after the execution of the program. You should consider potential process switches at the instruction level.

A load b      D load a
add 1         add 3
B store a     E store b
C print a     F print b

Note: “add” will not cause interference because it does not access shared memory
Note: P did not change b and Q did not change a, so only the order of the two prints matters, how C is ordered with DE and how F is ordered with AB does not matter

- ABDE CF/FC      2, 5 / 5, 2 -- Q loads a after P stores a
- ADEB/ADEB/DABE/DAEB CF/FC      2, 8 / 8, 2 -- two loads happens before stores
- DEAB CF/FC      8, 9 / 9, 8 -- P loads b after Q stores b

2. Consider the following program. Find a counter example (execution sequence) that demonstrates that this software solution is incorrect in terms of mutual exclusion violation.

```plaintext
var blocked: array[0..1] of boolean;
var turn: 0..1; // could be 0 or 1

function p (id: integer)
{ repeat
    blocked[id] := true;
    while (turn != id) do
        { while blocked[1-id] do; // do nothing
            turn := id;
        }
    < critical section >
    blocked[id] := false;
    < remainder code for p>
    until false;
}

main()
{ blocked[0] := false; blocked[1] := false;
  create thread to execute P(0);
  create thread to execute P(1);
}
```
P1: set blocked[1] = true;
P1: check and find turn is not 1 ( while (turn != id) do )
P1: check and find blocked[0] = false, exit inner loop ( while (blocked[1–id]) do; )
P0: set blocked[0] = true;
P0: check and find turn is 0, exit outer loop ( while (turn != id) do )
P0: enter critical section
P1: set turn = 1;
P1: check and find turn is 1, exit outer loop ( while (turn != id) do )
P1: enter critical section

3. Consider a sequential program as follows. Assume that we try to execute all statements concurrently to achieve maximal parallelism, but we want to obtain the same output as the sequential execution. Use (a) locks (b) semaphores to control the synchronization so that the program can be executed in maximal concurrency while still get the same outcome as the sequential execution.

1) x = y+z;   -- x should be computed before it is used in 4) and 5)  
2) w = y*z;   -- w should be computed first before it is used in 3) and 5)  
3) u = w–3;   -- u, v, k are not used in later statements, so does not matter  
4) v = x+t;  
5) k = x+w;

(a)
lock lx = true;  // locked, done before any execution  
lock lw = true;  // locked, done before any execution  
x := y+z;  unlock(lx);  
w := y*z;  unlock(lw);  
lock(lw); unlock(lw);  u = w–3;  
lock(lx); unlock(lx);  v := x+t;  
lock(lx); unlock(lx);  lock(lw); unlock(lw);  k = x+w;

(b)
semaphore sx = 0;
semaphore sw = 0;

x = y+z;  signal (sx); signal (sx);  
w = y*z;  signal (sw); signal (sw);  
wait(sw);  u = w–3;  
wait(sx);  v = x+t;  
wait(sx);  wait(sw);  k = x+w;

4. In the Jurassic Park Amusement center, tourists ride on a tour bus to visit the park. Each bus takes two visitors from the visitor center, drives around the park as long as the passengers wish, and return to the visitor center to drop off the passengers. There are N tour buses on service. If the N buses are all out, then a passenger who wants a ride must wait. If a bus is ready to load but there are no waiting visitors, then the bus waits. A bus does not leave until it loads two passengers. A bus does not load passengers if there is another bus that is only partly filled.

Consider the problem of synchronizing the bus processes and the visitor processes using semaphores. The code for visitor process follows. Three semaphores cust, bus, and busReady are used. Write the code for the bus process that performs the proper operations on the three semaphores. Use extra semaphores as needed. In the bus process, you can use the same pseudo-statements “visitor get in bus”, etc. as in the visitor
process. Do not forget to properly initialize the semaphores (including the three given in the code and any new ones you may introduce).

```pascal
var cust: semaphore := ?;  
    bus: semaphore := ?;  
    busReady: semaphore := ?;  

procedure visitor;  
begin  
    signal (cust);  
    wait (bus);  
    visitor gets in bus;  
    wait (busReady);  
    drive around the park;  
    visitor gets off bus;  
end;
```

```pascal
var cust: semaphore := 0;  
    bus: semaphore := 0;  
    busReady: semaphore := 0;  
    mutex := 1;

procedure bus;  
begin  
    wait (mutex);  
    signal (bus);  
    signal (bus);  
    wait (cust);  
    wait (cust);  
    signal (busReady);  
    signal (busReady);  
    signal (mutex);  
    drive around the park;  
    drop visitors;  
end;
```

```pascal
*** a bus needs to wait for two customers in mutex, otherwise, two waiting customers may get on two different buses.
```

5. Implement a monitor solution for the bakery problem. Assume that you have N salesmen and customers arrive at arbitrary times. Note: neither the customers nor the salesmen should have a busy waiting.

```pascal
Monitor  
{  
    int sales_count, initialized to N  
    int cust_count, initialized to 0  
    condition sales, cust;

    get_service()  
    {  
        if sales_count <= 0 sales.wait;  
        sales_count = sales_count - 1;  
        cust_count = cust_count + 1;  
        if cust_count < 1 cust.signal;  
    }
```
release_service()
{ } -- this can be omitted

prepare_service()
{ if cust_count <= 0 cust.wait;
  cust_count = cust_count - 1;
}

complete_service()
{ sales_count = sales_count + 1;
  if sales_count = 1 sales.signal; // sales_count = 1 => no sales were available
    // what happens if we remove the condition, always signal
}
} // end monitor

customer
{ get_service();
  receive the service
  release_service();
}
salesman
{ prepare_service();
  provide the service
  complete_service();
}

6. When we implement the code for realizing the monitor, there is one important issue. Monitor can only allow one active thread in it. When signaling a condition variable, we need to decide whether to let the signaler or the signalee to continue. We have discussed how to implement the monitor with the choice of letting the signaler continue. Now, discuss the high-level implementation idea if we choose to let the signalee continue.

Assume that the monitor is protected by a semaphore “mutex”. Either the signaler or the signalee continues, we need to maintain a counter “temp_count” and have a semaphore “temp” to hold the one that has to wait.

If the signalee proceeds and the signaler waits, we need to let the signaler do “temp.wait”.

When the signalee finishes using the monitor or when it goes for cond.wait on some condition, it needs to do the following:
  If temp-count > 0, temp.signal; else signal mutex;

7. Consider the bakery problem, but we replace the bakery by the hair salon. There are N hair dressers working for the salon. The customer can get the hair styling service if a hairdresser is available. The salon also has an entertainment center which has M seats. A customer can wait in the entertainment center for a hairdresser to become available. After finishing hair styling service, the customer makes payment to the salon. A salon manager process coordinates the customers and hairdressers. It can admit at most N+M customers. A customer, when wanting to have the hair styling service, sends a message to the salon manager. The manager “receives” customers from the “custport” and place them in the entertainment center. If there is a waiting customer, the manager “receives” an available hairdresser from the “dresserport” and pairs the
customer and the hairdresser for service. If there is no waiting customer, the manager should not receive a hairdresser (hairdressers wait in a room). After the hair styling service finishes, the customer should make payment via the “payport”. The customer and hairdresser processes are given as follows. Give the code for the salon manager process using “Guarded Communication”.

Customer process:
send (custport, customer);
take a seat in the entertainment center and wait;
get service from the hairdresser;
send (payport, payment);

Hairdresser process:
\[\text{repeat select} \]
send (dresserport, hairdresser);
provide hair styling service;
\[\text{until false;}\]

Manager process:
numcust = 0;
umserv = 0;

\[\text{repeat select} \]
\[\text{when numcust} < M \]
{ receive (custport, customer);
  numcust = numcust + 1;
  give the customer a seat in the entertainment center;
}
\[\text{when numcust} > 0 \text{ and numdresser} > 0 \] // gray part: can be omitted
{ receive (dresserport, hairdresser);
  pair customer and dresser;
  numcust = numcust – 1;
  numserv = numserv + 1;
  numdresser = numdresser – 1;
}
\[\text{when numserv} > 0 \]
{ receive (payport, payment);
  process payment;
  numserv = numserv – 1;
  numdresser = numdresser + 1;
}
\[\text{until salon-close-time;}\]

8. Consider a memory system with 1GB space (=1024MB). Currently the memory free list contains ((128MB, 64MB), (320MB, 8MB), (512MB, 16MB), (768MB, 256MB)) where each item (x, y) in the list represents the starting location x and length y of a free partition. Four jobs J_1, J_2, J_3, and J_4 of sizes 14MB, 20MB, 160MB, and 64MB, respectively, are to be loaded to the memory. Consider the first fit allocation policy and answer the following questions.
(a) Consider each job $J_i$, $1 \leq i \leq 4$. When $J_i$ is running, what would be the value for the base register and bound register?

J1 gets (128MB, 64MB) slot
⇒ base register 0x08000000, bound register 0x08dfffff
(or 0x08e00000 if we use A>=B as the circuit, but consider the above as the correct answer)

J2 gets allocated after J1 in (128MB, 64MB) slot
⇒ base register 0x08e00000, bound register 0x0a1fffff (starting address of next slot is 0x0a200000)

J3 gets (768MB, 256MB) slot
⇒ base register 0x30000000, bound register 0x39ffffff (starting address of next slot is 0x3a000000)

J4 gets allocated after J3 in (768MB, 256MB) slot
⇒ base register 0x3a000000, bound register 0x3dffffff (starting address of next slot is 0x3e000000)

(b) What is the free list after loading all four jobs?
((162MB, 30MB), (320MB, 8MB), (512MB, 16MB), (992MB, 32MB))

(c) Now $J_4$ is running and it accesses the logical address 0x04213a0c. Compute the corresponding physical address. Also, check and determine whether this causes access violation.

$J4$ base register 0x3a000000 + logical address 0x04213a0c
⇒ physical address 0x3e213a0c > bound register 0x3dffffff
⇒ access violation

(d) Now $J_4$ is running and it accesses the logical address 0x02f3f12c. Compute the corresponding physical address. Also, check and determine whether this causes access violation.

$J4$ base register 0x3a000000 + logical address 0x02f3f12c
⇒ physical address 0x3cf3f12c < bound register 0x3dffffff
⇒ good address, no access violation

(e) Answer the questions (a) through (d) for the best fit allocation policy.

For (a)

J1 gets (512MB, 16MB) slot
with base register: 0x20000000 and bound register: 0x20dffffff

J2 gets (128MB, 64MB) slot
with base register: 0x08000000 and bound register: 0x093ffffff

J3 gets (768MB, 256MB) slot
⇒ base register 0x30000000, bound register 0x39ffffff

J4 gets allocated after J3 in (768MB, 256MB) slot
⇒ base register 0x3a000000, bound register 0x3dffffff

For (b)
(148MB, 44MB), (320MB, 8MB), (526MB, 2MB), (992MB, 32MB)

For (c) and (d) because $J_4$ got the same allocation, so the answer is the same as before
9. Consider a 1GB memory system using buddy scheme. The system loads jobs \( J_1, J_2, J_3, J_4, \) and \( J_5 \) of size 40MB, 20MB, 160MB, 64MB, and 100MB, respectively.

(a) Compute the starting addresses of each job.
\( J_1 \) starting at 0x00000000, internal fragmentation = 24MB
\( J_2 \) starting at 0x04000000 (64MB), internal fragmentation = 12MB
\( J_3 \) starting at 0x10000000 (256MB), internal fragmentation = 96MB
\( J_4 \) starting at 0x08000000 (128MB), internal fragmentation = 0
\( J_5 \) starting at 0x20000000 (512MB), internal fragmentation = 28MB

(b) Give the list of free partitions. For each, include its starting address and size.
free list: ((96MB, 32MB), (192MB, 64MB), (640MB, 128MB), (768MB, 256MB))

(c) What is the total internal fragmentation?
Total fragmentation = 24+12+96+28 MB = 160MB

10. Consider a 256MB memory system using simple paging scheme. Each page is of size 16KB. The memory is byte addressable. The page table for Process \( P \) is as follows. Process \( P \)'s address space is only 5 pages. The numbers in the table are all in decimal.

\[ \begin{array}{c|c}
0 & 11 \\
1 & 25 \\
2 & 40 \\
3 & 3 \\
4 & 13 \\
\end{array} \]

(a) How many bits are required to address the entire memory?
(b) How many bits are required to address the offset within each page?
(c) What is the maximal possible internal fragmentation \( P \) can have?
(d) Given a logical address 0x0000ef5b, compute the corresponding physical address. Also, check and determine whether this causes access violation.
(e) Given a logical address 0x00017f5b, compute the corresponding physical address. Also, check and determine whether this causes access violation.

(a) 28 bits
(b) 14 bits
(c) 16KB – 1
(d) 0x0000ef5b = 1110 1111 0101 1011 \( \Rightarrow \) page 3, mapped to frame 3,
\( \Rightarrow \) physical address is the same as the logical address
(e) 0x00017f5b = 0001 0111 1111 0101 1011 \( \Rightarrow \) page 5, does not exist \( \Rightarrow \) access violation

11. Consider a virtual memory system with cache, main memory, and disk. The cache size is 256KB and has an access time 15ns. The cache hit ratio is 90% (which is considered low). The main memory is 256MB and has an access time 150ns. 99.99% of the memory accesses would find the demanded pages in the main memory and 0.01% of them would generate page faults. The disk size is 20GB and has an access time 1ms. Compute the average memory access time.

Average memory access time
\[
= (0.90 \times 15ns) + [(1-0.9) \times 0.9999] \times (150 + 15)ns + \\
\quad [(1 - (0.9 + (1-0.9) \times 0.9999))] \times (10^6 + 150 + 15)ns \\
= 1 \times 15ns + 0.1 \times 150ns + 0.00001 \times 10^6ns = 40ns
\]
12. Consider a virtual memory system using 32-bit addressing. The physical memory is of size $1\text{GB}$. Each page is of size 8KB. Assume that each entry of the page table fits in one word. A two-level page table is used and the root level contains 1K entries. A process P has the following page table:

<table>
<thead>
<tr>
<th>Root level page table</th>
<th>Level-2 page table starting at 0x00000000</th>
<th>Level-2 page table starting at 0x00000100</th>
<th>Level-2 page table starting at 0x00001000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00001000</td>
<td>Frame 150</td>
<td>Frame 33</td>
<td>Frame 160</td>
</tr>
<tr>
<td>0x00000100</td>
<td>Frame 21</td>
<td>disk</td>
<td>Frame 512</td>
</tr>
<tr>
<td>null</td>
<td>disk</td>
<td>Frame 120</td>
<td>....</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

(a) How many bits are required to address the root page table?
(b) How many bits are required to address each second level page table?
(c) Convert the logical address 0000 0000 0100 0000 0101 0000 1111 0101 (binary) to physical address.

(a) 10 bits
(b) 9 bits
(c) 0000 0000 0100 0000 0101 0000 1111 0101

It is root table entry 1, secondary page table entry 2, which is mapped to frame 120 = 0x78

Physical address is: 0000 0000 0000 1111 0001 0000 1111 0101