I.Introduction to operating systems.

1. The first INTEL family of CPUs (e.g. 8086/8088) did not include a hardware mode bit, and consequently both the operating system (e.g. DOS) and the user programs were executing in the same operational mode.

(a) Explain a problem that could occur because of the missing hardware mode bit.

- *A user program could accidentally or intentionally access the operating system resources without restriction. Consequently, a possible fatal crash of the operating system could occur.*

(b) Would it be possible for the operating system to implement a software mode bit in order to overcome that problem? Explain.

- *The hardware mode bit is used mostly to restrict access to some CPU resources (e.g. instructions, registers,...) by the user programs. Access to restricted CPU resources could possibly allow a program to alter other resources. For instance, an instruction could change the timer value, an instruction could allow to directly access a peripheral device, or access to the base register could change an address space.*
  A software mode bit should work similarly to a hardware mode bit, and should allow a user program to access only a permitted set of CPU resources. The implementation of the software mode bit would require the operating system to validate each CPU operation in order to check that it is legal in the context. Such implementation is not very suitable because it imposes too much overhead to achieve and consequently would reduce greatly the overall system performance.

2. Most modern operating systems implement a micro-kernel that contains less components than a conventional kernel. The kernel components that are not part of the micro-kernel have been moved to user space.

- Explain an advantage of the use of a micro-kernel instead of a conventional kernel.

- *A micro-kernel is smaller and thus requires less system resources and can be more reliable.*

II.Process management.

3. Explain which fields in the following process descriptor would not be required in a thread descriptor.

Field	Description
Process identification	An integer number to identify the process.
Process state	The current state of the process.
Owner	The identification of the user that created the process.
Parent process descriptor	A pointer to the descriptor of the parent of the process.
List of child process descriptors	A pointer to the list of processes that have been created by the process.
List of reusable and consumable resources	A pointer to the list of different resources allocated to the process, and for each resource the number of units allocated is accounted.
List of file descriptors	A pointer to the list of files that have been open by the process.
Message queue	A pointer to the message queue of the process.
Protection domain	A description of the access rights held by the process.
Processor registers content	The content of the program counter, status, base, limit and relocation registers for a process when it exited the running state for the last time.
General registers content	The content of the general registers for a process when it exited the running state for the last time.
Accounting information	A record of any statistical information about the process.
Scheduling information	A record of any information needed to schedule the process (e.g. priority, pointer to the scheduling queue, …).
Stack pointer	A pointer to the stack allocated to the process.

- *Process identification.*
  - *A thread also needs to be uniquely identified like a process.*

- *Process state.*
  - *A thread can also be in different states but its number of states is limited compared to the process.*

- *Owner.*
  - *The owner of a thread is by the default the owner of its controlling process, and consequently that field is only present in a process descriptor.*

- *Parent process descriptor.*
  - *The parent of a thread is by the default the parent of its controlling process, and consequently that field is only present in a process descriptor.*
  - *Alternately, a thread descriptor could use that field to identify its controlling process.*

- *Child process descriptors.*
  - *As a thread is created on behalf of its controlling process, that field is not required.*

- *List of resources.*
  - *As a thread is sharing most of the resources of its controlling process, that field is not required.*

- *List of file descriptors.*
  - *As a thread is sharing most of the open files of its controlling process, that field is not required.*

- *Message queue.*
  - *Messages are software resources that are handled by the controlling process but shared by the thread.*

- *Protection domain.*
  - *A protection domain applies to a set of resources that are shared by the threads of a controlling process.*

- *Processor registers.*
  - *A thread has a context that is different than the context of the controlling process, and therefore the processor registers are required.*

- *General registers.*
  - *A thread has a context that is different than the context of the controlling process, and therefore the general registers are required.*

- *Accounting information.*
  - *It may be required to maintain a few statistics about a thread.*

- *Scheduling information.*
  - *A thread may be scheduled differently than its controlling process, and therefore the scheduling information is required.*

- *Stack pointer.*
  - *A thread needs to have a stack to store its local variables.*

4. Consider a process manager that implements both processes and threads. The process represents the framework and the threads are used solely to execute the instructions. Among the following list of states, explain which ones should be more appropriate for a thread and for a process.

(i) New.

- *A thread and a process need to be created and therefore this state applies to both.*

(ii) Terminated.

- *A thread and a process need to be terminated and therefore this state applies to both.*

(iii) Running.

- *Only a thread can be set in execution and therefore this state only applies to threads.*

(iv) Ready.

- *Only a thread can be waiting to move into the execution state and therefore this state only applies to threads.*

(v)Waiting.

- *Only a thread can be blocked waiting for an I/O while it was in the running state and therefore this state applies only to threads.*

(vi) Swapped.

- *Only a process can be in the swapped state because the address space is allocated to it and not to the threads.*

5. Specify 3 events that may cause the dispatcher to be called.

- *Execution of a yield(…) or sleep(…) system call.*
- *Occurrence of an interrupt.*
- *Occurrence of a timer interrupt.*

6. In most operating systems, a process will be often blocked when it performs an I/O operation. (i) Provide a strategy that would allow a process to overlap its CPU and I/O operations. (ii) Can the process always overlap CPU and I/O operations?

(i) Use concurrent threads to perform I/O and CPU operations.
(ii) The overlap depends whether the I/O results is required by the CPU operations.

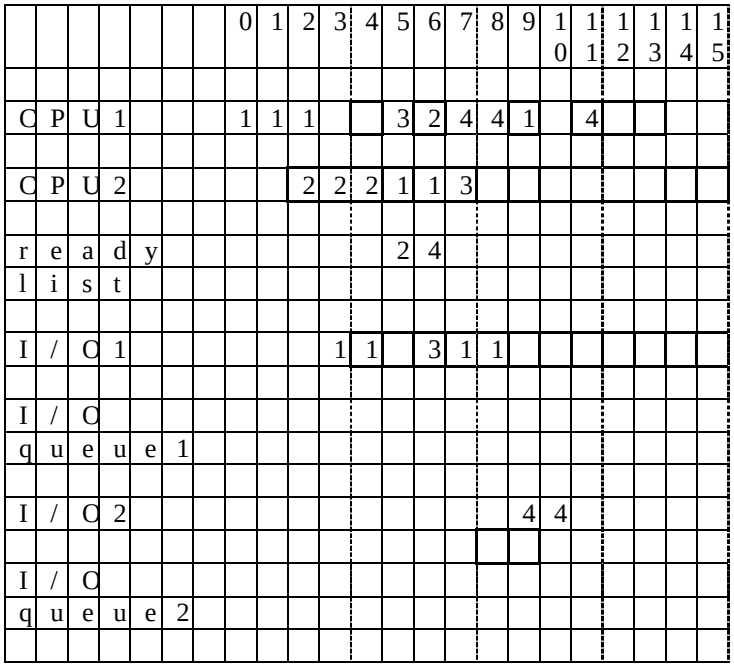
III.Process scheduling.

7. Consider a system having the following characteristics :

- A process load of four processes (i.e. P1, P2, P3, P4) whose characteristics are described in the table below. Assume that the ready list is empty when the first of the four processes starts its execution cycle.
- 2 CPUs (i.e. CPU1 and CPU2) that share the same ready list. When both CPUs are free, the next process is dispatched on CPU1 unless the process was already running on CPU2 when its time quantum expired.
- 2 I/O devices (i.e. I/O1 and I/O 2), I/O1 is used by P1, P3 and I/O2 is used by P2, P4.
- Round-robin scheduling algorithm with a time quantum of 3 units.
- Context switch time is infinitely small (i.e. 0).
- When a CPU becomes free, the next process is selected from the ready list. If the ready list is empty or there are no new processes then a process that has just ended its I/O operation has priority. Otherwise, the process that just ended its time quantum can hold on the CPU.

Process	Arrival time	Service time	I/O duration	I/O periods
P1	0	6	2	3, 5
P2	2	4	0	_
P3	5	2	1	1
P4	6	3	3	2

(a) Show the allocation state for the two CPUs, the ready list, the two I/O devices, and the I/O waiting queues.



(b) Calculate the average turnaround time and waiting time.

- *Turnaround time*
  $T_t = (10 + 5 + 3 + 6)/4 = 24/4 = 6.00$
- *Waiting time*
  $T_w = (0 + 1 + 0 + 1)/4 = 2/4 = 0.50$

8. In a round-robin scheduling, a long time quantum tends to increase the overall waiting time and to decrease the overall turnaround time. Explain a factor that causes the decrease of the turnaround time.

- *The long time quantum decreases the number of context switches and consequently the turnaround time is also decreased.*

9. Consider the following classes of processes :
   - Interactive.
   - Batch.

- Kernel.
- Multimedia.

and the following scheduling algorithms :

- First come first served.
- Shortest job next.
- Nonpreemptive external priority.
- Deadline.
- Round-robin.
- Shortest remaining time first.
- Preemptive internal priority.
- Multilevel queues.

- Associate each class of processes to an appropriate scheduling algorithm.

- *Interactive : Shortest job next.*
- *Batch : First come first served.*
- *Kernel : Nonpreemptive external priority.*
- *Multimedia : Deadline.*

10. Some operating systems associate a higher priority to I/O-bound processes in order to improve the overall performance of the system.

- Explain which performance measure (e.g. waiting time or response time) that will be improved.

- *The response time will be improved because the I/O-bound processes have short CPU bursts.*

IV.Process synchronization and communication.

11. Modify the following code for the Peterson's algorithm so that it works with n processes instead of 2. The processes that first set their flags to TRUE must be selected first to enter their critical section.

- Shared variables.
  bool flag[2]=false,
  turn=0; // Or turn=1

– Process Pi.

```
…
flag[i] = true;
turn = j;
while (flag[j] && turn == j);
// Critical section
flag[i] = false;
```

```
// Remainder section
…
```

– Process Pj.

```
…
flag[j] = true;
turn = i;
while (flag[i] && turn == i);
// Critical section
flag[j] = false;
// Remainder section
…
```

- Shared variables.

n = ...; // Number of processes p1, p2, …, pn
bool flag[***n***] = {FALSE}; // Set all flags to FALSE
int turn=***-1***; // Set turn to nobody

- Process Pi.

```
…
number[i] = getNumber();    // Get a number to wait in line
flag[i] = TRUE;
if (turn == 0)      // If it is nobody's turn, take the turn
    turn = i;

while (turn != i); // Wait for turn
// Critical section
flag[i] = FALSE;
curNumber = number[i];  // Current process in critical section
number[i] = 0;              // Return the ticket
next = i;                   // Choice of nex process
j =( i + 1)%n;  // Point to next process
found = false;
while ( ( j!=i ) and !found)    // Find next process with a flag set to TRUE
                                // and having the next ticket number
{   if (flag[j])
        if (number[j] == (curNumber +1))
        {   next = j;
            found = true;
        }
    j=(j+1)%n;
}
if (next == i)  // Set turn to 0 or to next process
    turn = 0;
else
    turn = next;
```

```
// Remainder section
…
```

12. Consider the following solution to the readers and writers problem :

- Reader and writer processes compete for the access to a shared resource.
- Only one writer can be in its critical section whereas many readers can share the critical section.
- When a reader is in its critical section, all writers must wait until there are no more read requests. In other words, once a reader has acquired the critical section all subsequent readers have priority over the writers. Thus, when the last reader ends it will wake-up a waiting writer.
- When a writer is in its critical section, all readers must wait until there are no more write requests. In other words, once a writer has acquired the critical section all subsequent writers have priority over the readers. Thus, when the last writer ends it will wake-up a waiting reader.

- Show the synchronization code for the reader and writer processes using semaphores.

- *Reader*

```
...
wait(mutex1);
readcount++;
if (readcount == 1)
    wait(access);
signal(mutex1);
//  Critical section
wait(mutex1);
readcount--;
if (readcount == 0)
    signal(access);
signal(mutex1);
    ...
```

- *Writer*

```
...
wait(mutex2);
writecount++;
if (writecount == 1)
    wait(access);
signal(mutex2);
wait(write);
//  Critical section
signal(write);
wait(mutex2);
```

```
writecount--;
if (writecount == 0)
    signal(access);
signal(mutex2);
    ...
```

13. Suppose that the mailboxes used for interprocess communication are allocated directly into the address space of each process. (i) Would the mailboxes be allocated by the compiler or by the kernel? Explain. (ii) What problem this strategy may cause?

(i) Compiler.
(ii)User program could inadvertently overwrite the mailbox.

V.Deadlock.

14. Show that the current allocation state is safe using the Banker's algorithm.

Resources (Rj)	Units (Cj)
R0	3
R1	4
R2	6
R3	2

- Count of resources

Process (Pi)	R0	R1	R2	R3
P0	2	1	3	1
P1	0	2	3	2
P2	1	0	2	1
P3	0	2	4	0
P4	1	2	3	1

- Maximum claims

Process (Pi)	R0	R1	R2	R3
P0	1	0	1	1
P1	0	2	1	0
P2	1	0	0	0
P3	0	0	1	0
P4	1	1	2	0
Sum	3	3	5	1

- Current allocation state

- *Sequence of execution*

  - *P4*
  - *P0*
  - *P1*
  - *P2*
  - *P3*

- *The current allocation state is safe because there exists a sequence for which all processes could be executed when claiming their maximum needs of resources.*

15. Show that the current allocation state is deadlocked or not using a resource allocation graph.

Resources (Rj)	Units (Cj)
R0	2
R1	3

- Count of resources

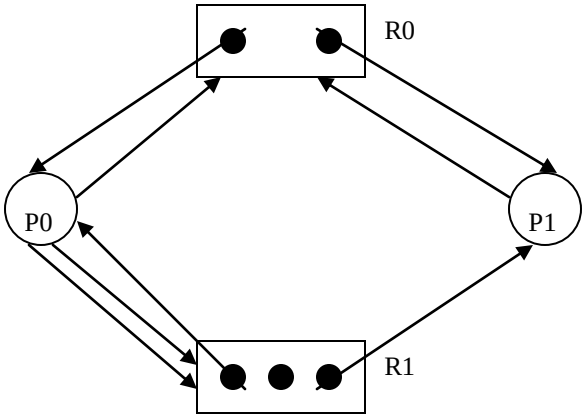
Process (Pi)	R0	R1
P0	1	1
P1	1	1
Sum	2	2

- Current allocation state

Process (Pi)	R0	R1
P0	1	2
P1	1	0

- Current requests

- *Initial system state.*



- *The current allocation state is deadlocked because none of processes P0 and P1 can execute, and therefore the graph can not be fully reduced.*

VI.Memory management and virtual memory.

16. Consider a virtual memory system with the following characteristics :

- Page and frame sizes of 1 Kbyte.
- Physical memory size of 8 Gbytes.
- Virtual memory (i.e. swap space) size of 2 Gbytes.
- Use of the paging technique.
- Use of a static page allocation algorithm.
- Use of the LRU replacement algorithm.

(a) Show the format of a virtual address and of a physical address by specifying the number of bits for the segments, pages, frames, and offsets.

- *Virtual address*

21	10
p	*d*

- *Physical address*

23	10
f	*d*

(b) Consider a process having an address space of 32 Mbytes. Calculate the physical address for an instruction that is in logical address 10728. The logical address is loaded into frame 4096 in physical memory.

- *The page for the logical address 10728 is ⌊10728/1024⌋= 10.*
- *The displacement within page 10 is 10728%1024 = 488.*
- *The physical address of frame 4096 is 4096*1024 = 4 194 304*
- *The physical address of the logical address 10728 is 4 194 304 +488 = 4 194 792.*

(c) Consider the following page reference string R={0, 1, 0, 1, 2, 3, 4, 2, 3, 4, 5, 6, 7, 8} for the process described in (b).

i. Show the memory representation for the LRU algorithm. Assume an allocation of 3 frames.

Frame	0	1	0	1	2	3	4	2	3	4	5	6	7	8
0	***0***	0	0	0	0	***3***	3	3	3	3	3	***6***	6	6
1		***1***	1	1	1	1	***4***	4	4	4	4	4	***7***	7
2					***2***	2	2	2	2	2	5	5	5	***8***

ii. Calculate the page fault rate

- Page fault rate is 9/14 = 0.64

17. Consider the page reference string ω={0, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 7 } for a given process. Show the memory representation of the pages using the working set model with a window size Δ=2.

	0	***1***	***2***	***3***	***4***	***5***	***4***	***5***	***6***	***7***	***8***	***7***
0	***0***	***0***	***2***	***2***	***4***	***4***	***4***	***4***	***6***	***6***	***8***	***8***
1		***1***	***1***	***3***	***3***	***5***	***5***	***5***	***5***	***7***	***7***	***7***
Alloc	***1***	***2***	***2***	***2***	***2***	***2***	***2***	***2***	***2***	***2***	***2***	***2***

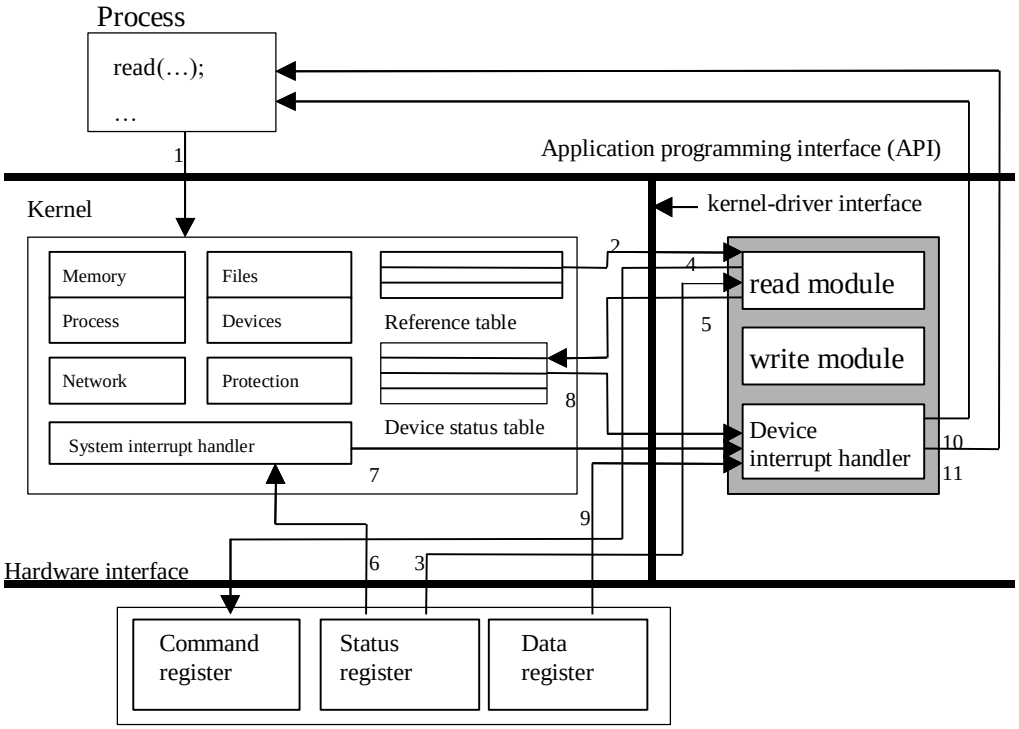
18. Consider the instruction MOVE X,Y where X is a destination memory location and Y is a source memory location. The instruction is in page 1, X is in page 7, and Y is in page 8. How many page faults would be generated?

- *The instruction could generate up to 3 page faults when it is executed.*

VII.I/O and file system.

19. The following diagram shows the steps for a read operation using the interrupt-driven I/O technique.

- Specify all the steps for a write operation using the diagram.



- *The process makes a system call to **write** data to a device.*
- *The kernel authenticates the system call and executes a trap instruction (i.e. software interrupt).*
- *The kernel activates the **write module** of the driver using a pointer in the reference table.*
- *The top half of the driver periodically queries the status register to determine if the device is idle, and waits for it to be ready.*
- ***The driver copies the data from the user process into the data register of the controller.***
- *The driver stores an **output** command into the command register of the controller, thereby starting the device.*
- *The top half of the driver saves information regarding the current operation in the device status table.*
- *The device manager invokes the process scheduler to execute another process.*
- *When the device has completed the **write** operation, it raises an interrupt by asserting a signal on the interrupt request line of the CPU.*
- *The CPU dispatches the system interrupt handler to identify the device that caused the interrupt.*
- *The system interrupt handler passes execution control to the concerned device interrupt handler using the vector table.*
- *The device interrupt handler (i.e. bottom half of the driver) retrieves the pending I/O status information from the device status table.*

– *The device interrupt handler returns execution control to user the process.*

20. Consider a file system with the following characteristics :
    - A single index block that is pointed to by a directory entry.
    - Block size of 8 Kbytes.
    - Pointer size of 32 bits.

(a) Calculate the maximum size for a file.

- *8 K / 4 = 2048 pointers to data blocks in the index block.*
- *Maximum file size is 2048x8K = 16 M*

(b) Suppose that the file system allows the index blocks to be linked linearly, calculate the maximum size for a file.

- *2047 pointers to data blocks in each index block.*
- *1 pointer to the next index block in each index block.*
- *2^{32}+1= 4 G+1 index blocks.*
- *Maximum file size is ((4 G+1)x(2047x8K))*

21. Consider a disk system with a transfer rate of 5 MB/sec and a process having a size of 256 KB. Calculate the time required to swap that process entirely.

- *256 K / 5 M = 51.2/K = 0.05 secs = 50 ms*

22. Could a file system span across two distinct disk drives or partitions? If yes explain how, if no explain why.

- *A same file system could span across multiple disk partitions as long as each block pointer identifies also the partition in addition to the block address.*

VIII.Protection.

23. A protection domain can be assigned statically or dynamically to a process. In a static assignment the process uses the same domain during its lifetime whereas it can switch domains in a dynamic assignment.

- Consider an operating system that implements the user and kernel modes of operations. Would you implement static or dynamic domain assignment to processes? Explain.

- *A dynamic domain assignment is required because the protection domain of the process also changes when it switches from user mode to system mode and vice-versa.*

24. Explain (i) a difference that exists between a capability list and an access list and (ii) an advantage that an access list has over a capability list.

(i) The access list specifies the access rights owned by an object in all possible domains whereas a capability list specifies the access rights for a domain on all possible objects.
(ii) Users can control the access rights on their objects.

25. Explain the way that the system restricts the user applications from directly accessing the I/O devices.

- *The I/O machine instructions are privileged and can be executed only in the supervisor mode.*

26. Consider a computer system that has no hardware support for relocation (e.g. MMU, relocation registers, TLB, ...). Could it be possible to implement some kind of memory protection in such system? Explain.

- *The operating system could implement a software support for memory protection that would work similarly to the hardware support.*
  A pair of base and limit registers would be maintained in the user program memory, and for each memory reference a validation routine would be automatically called.