

Université d'Ottawa  
Faculté de génie

École d'ingénierie et  
de technologie de l'information



University of Ottawa  
Faculty of Engineering

School of Information  
Technology and Engineering

## CSI 3131 Operating Systems

### FINAL EXAM

**Length of Examination: 3 hours**

**April 29, 2010, 14:00**

**Professor: Gilbert Arbez**

Name: \_\_\_\_\_

Student Number: \_\_\_\_\_

Closed book, no scientific calculators or other electronic devices (except simple calculators) are allowed.

If you do not understand a question, clearly state an assumption and proceed.

At the end of the exam, when time is up:

- Stop working and turn your exam upside down.
- Remain silent.
- Do not move or speak until all exams have been picked up, and a TA or the Professor gives the go-ahead to leave.

**The exam consists of three (3) parts**

<b>Part 1</b>	<b>Short Answer Questions</b>	<b>10 points</b>
<b>Part 2</b>	<b>Theory Questions</b>	<b>20 points</b>
<b>Part 3</b>	<b>Problem Solving Questions</b>	<b>30 points</b>
<b>Total</b>		<b>60 points</b>

Number of pages: 14

**Short Answer Questions – (5 X 2 points = 10 points total)**

Answer the following questions in the provided space. Answer with a few words or a single phrase.

- 1) What type of variable are used in "monitors" to support synchronization of processes/threads?
  
  
  
  
  
  
  
  
  
  
- 2) What file allocation method is used in the FAT file system?
  
  
  
  
  
  
  
  
  
  
- 3) Name at least 2 places in the OS where FCFS (first come first served) algorithm can be used.
  
  
  
  
  
  
  
  
  
  
- 4) Give at least one advantage of virtual memory management over the first memory management schemes?
  
  
  
  
  
  
  
  
  
  
- 5) How does a UNIX operating system allow communication between two processes running on different physical computers?

## Part 2 – Theory Questions (20 points)

1. (5 points) Complex disk scheduling algorithms have better performance than simpler algorithms, but performance difference varies with different request loads. Associate each of the following algorithms to one (and only one) the following load conditions: FCFS (First-Come First Serve), SSTF (Shortest Seek Time First), C-LOOK (look version of the circular scan). Explain your answer.

Lightly loaded disk (0-3 requests in the scheduling queue):

Medium loaded disk (0-20 requests in the scheduling queue):

Heavily loaded disk (30-50 requests in the scheduling queue):

2. (5 points) The page fault rate (page faults per second) can be used as part of a resident set management policy. Explain the circumstances under which such a policy will suspended/swapped out a process?

3. (5 points) Explain how the TLB can be used to increase the performance of the OS?
4. (5 points) Paging memory management can be improved with page buffering; explain how page buffering is applied and how it improves the memory management?

### Part 3 – Problem solving questions (30 points)

#### Question 1 - Deadlock Avoidance (5 points)

The following shows the state of 5 processes (P0 to P4) sharing 3 resources (R0 to R2).

$$\text{Max} = \begin{matrix} & \begin{matrix} R0 & R1 & R2 \end{matrix} \\ \begin{matrix} P0 \\ P1 \\ P2 \\ P3 \\ P4 \end{matrix} & \left\{ \begin{matrix} 0 & 0 & 1 \\ 2 & 7 & 5 \\ 6 & 6 & 5 \\ 4 & 3 & 5 \\ 0 & 6 & 5 \end{matrix} \right\} \end{matrix} \quad \text{Allocated} = \left\{ \begin{matrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 0 & 3 \\ 2 & 3 & 5 \\ 0 & 3 & 3 \end{matrix} \right\} \quad \text{Available} = \{ 2, 1, 0 \}$$

- What is the total number of instances for each type of resource?
- Complete the Need matrix (i.e. show the maximum amount of resources each process might request).

$$\text{Need} = \left\{ \begin{matrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix} \right\}$$

- Show that the system is in a safe state by completing the following table (i.e. apply the safety algorithm).

Process	Need	Allocated	Work	Finished

- d. Can a request from P2 for the resources (0, 1, 0) be granted? Show how you arrived at your answer by applying the Banker's algorithm (please show all your steps). Indicate how the OS will treat the requesting process P2.

Available = { , , }

$$\text{Max} = \begin{matrix} & \begin{matrix} R0 & R1 & R2 \end{matrix} \\ \begin{matrix} P0 \\ P1 \\ P2 \\ P3 \\ P4 \end{matrix} & \begin{bmatrix} 0 & 0 & 1 \\ 2 & 7 & 5 \\ 6 & 6 & 5 \\ 4 & 3 & 5 \\ 0 & 6 & 5 \end{bmatrix} \end{matrix} \quad \text{Allocated} = \left\{ \begin{matrix} \\ \\ \\ \\ \end{matrix} \right\} \quad \text{Need} = \left\{ \begin{matrix} \\ \\ \\ \\ \end{matrix} \right\}$$

Process	Need	Allocated	Work	Finished

## Question 2 – Memory Management (10 points total)

### (a) Segmentation with Paging (5 points)

There is one process in a system that currently uses 3 segments; they contain the following (this is a simplified example; the strings represent variable size content, i.e. variable size segments):

Segment	Content
0	<i>Pointers_are_great_but_quite_a_challenge.</i>
1	<i>Data_can_exist_on_the_stack_or_heap_or_data_segment.</i>
2	<i>Stack_and_heap_are_important.</i>

The "simplified" system uses an 8-bit logical address with paged segments. Each process can have at most 4 segments, each paged with up to 4 pages; a page contains 16 bytes. Thus segments can be at most 64 bytes long (4 pages x 16 bytes) with the process at most 256 bytes long (4 segments x 64 bytes). The content of the physical memory is as follows (ignore the fact that there is no space for the OS itself and other processes):

Frame #	Frame content	Frame Description
0 (0000)	<i>Pointers_are_gre</i>	Data/code
1 (0001)	< , , , >	Segment table
2 (0010)	< , , , >	Page table of segment 0
3 (0011)	Old_data_here	Empty
4 (0100)	< , , , >	Page table of segment 2
5 (0101)	<i>Data_can_exist_o</i>	Data/code
6 (0110)	< , , , >	Page table of segment 1
7 (0111)	<i>heap_or_data_segmen</i>	Data/code
8 (1000)	<i>at_but_quite_a_c</i>	Data/code
9 (1001)	<i>hallenge.</i>	Data/code
10 (1010)	<i>n_the_stack_or_h</i>	Data/code
11 (1011)	<i>ent.</i>	Data/code
12 (1100)	xxxx_xxxx_xxxx	Empty
13 (1101)	<i>Stack_and_heap_a</i>	Data/code
14 (1110)	aaaa_bbbb_cccc	Empty
15 (1111)	<i>re_important.</i>	Data/code

- (2 points)** In the table above, fill in the empty fields in the segment table and in the page tables of the segments. Use "-1" to indicate an invalid segment/page (only 4 entries exist in each table).
- (1 point)** The logical address is 8 bits long as is the physical address. Give the number of bits for the following:
  - Offset into the frame/page:
  - Frame number:
  - Segment number:
  - Page number:

iii. (2 points) Fill in the last two columns of the following table:

Logical address	Corresponding physical address (or access violation)	Data/code byte at this address
00000101		
01110001		
10111110		
01101101		



**(b) Virtual Memory Management (5 points)**

The following table shows how 4 physical frames allocated to a process have been loaded with its logical pages (at time 164). All frame/page values are numbered using decimal values and starting at 0. All timing values correspond to the number of clock ticks from the start of the process execution. The U bit is the use bit that is set whenever the loaded page is referenced.

Page Frame Number	Virtual Page Number	Time loaded	Time Referenced	U Bit
0	2	60	161	0
1	1	130	160	1
2	0	26	162	1
3	3	20	163	1

For each page replacement algorithm below, circle the virtual page numbers that cause a page fault in the given sequence of virtual page accesses. Under each page number that causes a page fault, show the page replacement by giving the new list of loaded virtual pages (i.e. the new contents of the second column of the above table). It is NOT necessary to show the contents of the list for the virtual pages that do not cause a page fault.

Optimal      4      0      0      0      2      4      2      1      0      3      2

Clock      4      0      0      0      2      4      2      1      0      3      2

Assume that a pointer is initially set to Page Frame 2 at time 164.

LRU      4      0      0      0      2      4      2      1      0      3      2

### Question 3 - Semaphores (5 points)

Develop the synchronization logic of a unisex washroom simulation according to the following instructions:

1. Men and women cannot be in the washroom at the same time.
2. At most 3 people can be in the washroom (either 3 men or 3 women).
3. The method `useWashroom()` simulates a person using the washroom and called when the person has the right to enter and use the washroom according to instructions 1 and 2.
4. The method `maleUse()` is called by male "threads" to use the washroom; complete **this** method. (The method `femaleUse()` will be used by female "threads" and has the same logic.)
5. In the method, use the following semaphores and variable; give initial values for them in the declarations below.
  - a. Variable `maleNum` to denote the number of males that are allowed to use the washroom (either waiting for it because it is full, or already inside).
  - b. Semaphore `maleAccess` to control access to the variable (male threads will block on this Semaphore when women are in the washroom).
  - c. Semaphore `maleEnter` to control the number of males in the washroom (male threads will block on this Semaphore when 3 men are in the washroom).
  - d. Semaphore `empty` for gaining access to the washroom when the washroom is empty (this semaphore is used by both male and female threads)(Note, the method `femaleUse()` would use the variable `femaleNum`, the semaphores `femaleAccess` and `femaleEnter`; and also the semaphore `empty`).
6. Your solution must not deadlock, but do not worry about starvation.

// Semaphores/variable declarations: Show initial values

```
int maleNum = ____
```

```
Semaphore maleAccess=____, maleEnter=____, empty=____
```

```
maleUse()  
{
```

```
    useWashroom();
```

```
}
```

### Question 5 - File Systems (10 points)

The exam annex provides a description of the contents of the Quantum Mechanics<sup>1</sup> Minix File System including:

- A diagram that illustrates the organization of the directories and files in the Minix file system; directory names are in bold font and file names in regular font.
- A block allocation table that gives the numbers of the blocks (Minix zones) where the content of each directory and file is located; the order of the numbers reflect the order of the logical blocks.
- The listing produced by “ls -lR” executed from the root of the mounted Minix file system.

Use the available information in the annex to complete the Minix file system structures on the following pages: the inodes, directory tables and index blocks. It is NOT necessary to fill in the fields that are greyed out.

---

<sup>1</sup> The directory names are names of scientists who contributed to the development of quantum mechanics and the file names are related to subjects they studied.

# MINIX File System

MINIX Inode Table							
inode #	i_mode	i_uid	i_size	i_time	i_gid	i_nlinks	i_zone[0..8]
1 (root)	0100 000 101101101		128			2	524,0,0,0,0,0,0,0,0
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							

## MINIX Directory Tables

<b>Directory: <i>root</i></b>	
<b>Ino</b>	<b>Name</b>

<b>Directory: bohr</b>	
<b>Ino</b>	<b>Name</b>

<b>Directory: plank</b>	
<b>Ino</b>	<b>Name</b>

<b>Directory: deBroglie</b>	
<b>Ino</b>	<b>Name</b>

<b>Directory: einstein</b>	
<b>Ino</b>	<b>Name</b>

<b>Directory: schrodinger</b>	
<b>Ino</b>	<b>Name</b>

<b>Directory: born</b>	
<b>Ino</b>	<b>Name</b>

The following table can be used to define index blocks. You may use any data block (i.e. block number) not used in the Block Allocation Table of the annex (i.e. block allocated to content of directories and files. Data blocks start at block number 524.

<b>Index Blocks</b>	
<b>Index Block Number</b>	<b>Index Block Content</b>

Université d'Ottawa  
Faculté de génie

École d'ingénierie et  
de technologie de l'information



University of Ottawa  
Faculty of Engineering

School of Information  
Technology and Engineering

## CSI 3131 Operating Systems

### FINAL EXAM Solution

**Length of Examination: 3 hours**

**April 29, 2010, 14:00**

**Professor: Gilbert Arbez**

Name: \_\_\_\_\_

Student Number: \_\_\_\_\_

Closed book, no scientific calculators or other electronic devices (except simple calculators) are allowed.

If you do not understand a question, clearly state an assumption and proceed.

At the end of the exam, when time is up:

- Stop working and turn your exam upside down.
- Remain silent.
- Do not move or speak until all exams have been picked up, and a TA or the Professor gives the go-ahead to leave.

**The exam consists of three (3) parts**

<b>Part 1</b>	<b>Short Answer Questions</b>	<b>10 points</b>
<b>Part 2</b>	<b>Theory Questions</b>	<b>20 points</b>
<b>Part 3</b>	<b>Problem Solving Questions</b>	<b>30 points</b>
<b>Total</b>		<b>60 points</b>

Number of pages: 12

### **Short Answer Questions – (5 X 2 points = 10 points total)**

Answer the following questions in the provided space. Answer with a few words or a single phrase.

- 1) What type of variable are used in "monitors" to support synchronization of processes/threads?

**Conditional variables.**

- 2) What file allocation method is used in the FAT file system?

**Linked allocation.**

- 3) Name at least 2 areas in the OS where FCFS (first come first served) algorithm is available:

**CPU scheduling, Disk Scheduling, Page Replacement**

- 4) Give at least one advantage of virtual memory management over previous memory management schemes?

**Process memory space can be larger than physical space.**

**More processes can be loaded into physical memory – increase in multi-programming.**

- 5) How does a UNIX operating system allow communication between two processes running on different systems?

**Sockets.**



## Part 2 – Theory Questions (20 points)

1. (5 points) Complex disk scheduling algorithms have better performance than simpler algorithms, but performance varies with different loads. Associate one of the following algorithms to the following load conditions (only one and the most suitable; algorithms should only be allocated once): FCFS (First-Come First Serve), SSTF (Shortest Seek Time First), C-LOOK (look version of the circular scan). Explain your answer.

Lightly loaded disk system (0-3 requests in the scheduling queue):

**FCFS – For lightly loaded systems, all algorithms are equal, hence FCFS is suitable.**

Medium loaded disk system (0-20 requests in the scheduling queue):

**SSTF – Suitable for medium loaded; starvation does not occur since queue empties.**

Heavily loaded disk system (30-50 requests in the scheduling queue):

**C-LOOK – Required for heavily loaded system to ensure no starvation.**

2. (5 points) The page fault rate (page faults per second) can be used as part of a resident set management policy. Explain the circumstances under which such a policy will suspended/swapped out a process?

**The system monitors the page fault rate (# of page fault per fixed time slot) for each process. If the page fault rate is higher than an upper threshold, the resident set size of the process is increased, if the rate is lower than the lower threshold, the resident set size is decreased. If the resident set size cannot be increased (because all other processes also have high rate of page faults), a process is chosen and swapped out to free up pages to help reduce the page fault rates of other processes.**

3. (5 points) Explain how the TLB can be used to increase the performance of the OS?

**The translation lookaside buffer is a cache of page table entries using associative memory to find the logical page number. It is used to improve the efficiency of the memory management system by increasing the speed of translation of logical addresses to physical addresses; this is achieved by providing a faster lookup of page table entries – this is possible because of locality of reference.**

4. (5 points)? Paging memory management systems can be improved with page buffering; how is it used and why?

**Two lists are created to buffer pages when they are replaced, a free list and a modified list. When a page is selected for replacement, it is moved to either list (modified list if the page has been modified, and the free list otherwise) and the page at the head of the free list is used to contain the newly loaded page. When a page fault occurs, the lists are checked to see if the required page is present – this prevents the need to load the page from disk. This improves considerably the performance of page replacement algorithms such as FIFO by having access to pages that should have not been replaced.**

### Part 3 – Problem solving questions (30 points)

#### Question 1 - Deadlock Avoidance (5 points)

The following system is a snapshot of 5 processes (P0 to P4) sharing 3 resources (R0 to R2).

$$\text{Max} = \begin{matrix} & \begin{matrix} R0 & R1 & R2 \end{matrix} \\ \begin{matrix} P0 \\ P1 \\ P2 \\ P3 \\ P4 \end{matrix} & \left\{ \begin{matrix} 0 & 0 & 1 \\ 2 & 7 & 5 \\ 6 & 6 & 5 \\ 4 & 3 & 5 \\ 0 & 6 & 5 \end{matrix} \right\} \end{matrix} \quad \text{Allocated} = \begin{matrix} & \begin{matrix} R0 & R1 & R2 \end{matrix} \\ & \left\{ \begin{matrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 0 & 3 \\ 2 & 3 & 5 \\ 0 & 3 & 3 \end{matrix} \right\} \end{matrix} \quad \text{Available} = \{ 2, 1, 0 \}$$

- a. What is the total number of instances for each type of resource?

$$R1 = 6 \quad R2 = 7 \quad R3 = 12$$

- b. Complete the Needs matrix (i.e. show the amount of resources each process might need).

Simply Max – Allocated:

$$\text{Need} = \begin{matrix} & \begin{matrix} R0 & R1 & R2 \end{matrix} \\ & \left\{ \begin{matrix} 0 & 0 & 0 \\ 0 & 7 & 5 \\ 6 & 6 & 2 \\ 2 & 0 & 0 \\ 0 & 3 & 2 \end{matrix} \right\} \end{matrix}$$

- c. Show that the system is in a safe state by completing the following table.

Process	Need	Allocated	Work	Finished
			{2, 1, 0}	{F, F, F, F, F}
P0	{0, 0, 0}	{0, 0, 1}	{2, 1, 1}	{T, F, F, F, F}
P3	{2, 0, 0}	{2, 3, 5}	{4, 4, 6}	{T, F, F, T, F}
P4	{0, 3, 2}	{0, 3, 3}	{4, 7, 9}	{T, F, F, T, T}
P1	{0, 7, 5}	{2, 0, 0}	{6, 7, 9}	{T, T, F, T, T}
P2	{6, 6, 2}	{0, 0, 3}	{6, 7, 12}	{T, T, T, T, T}

- d. If a request from P2 for the resources (0, 1, 0) arrives, can it be granted? Show how you arrived at your answer by updating the state of the system, and completing the table below. Indicate how the OS will treat the requesting process P2.

$$(0,1,0) < \text{Need}[2] = (6,6,2)$$

$$(0,1,0) < \text{Available} = (2,1,0)$$

If request is accepted, are we in a safe state, apply safety algorithm:

$$\text{Available} = \{ \mathbf{2, 0, 0} \}$$

$$\text{Max} = \begin{matrix} & \begin{matrix} R0 & R1 & R2 \end{matrix} \\ \begin{matrix} P0 \\ P1 \\ P2 \\ P3 \\ P4 \end{matrix} & \begin{Bmatrix} 0 & 0 & 1 \\ 2 & 7 & 5 \\ 6 & 6 & 5 \\ 4 & 3 & 5 \\ 0 & 6 & 5 \end{Bmatrix} \end{matrix} \quad \text{Allocated} = \begin{Bmatrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 1 & 3 \\ 2 & 3 & 5 \\ 0 & 3 & 3 \end{Bmatrix} \quad \text{Need} = \begin{Bmatrix} 0 & 0 & 0 \\ 0 & 7 & 5 \\ 6 & 5 & 2 \\ 2 & 0 & 0 \\ 0 & 3 & 2 \end{Bmatrix}$$

Process	Need	Allocated	Work	Finished
			{2, 0, 0}	{F, F, F, F, F}
P0	{0, 0, 0}	{0, 0, 1}	{2,0,1}	{T, F, F, F, F}
P3	{2, 0, 0}	{2, 3, 5}	{4, 3, 6}	{T, F, F, T, F}
P4	{0, 3, 2}	{0, 3, 3}	{4, 6, 9}	{T, F, F, T, T}

**Not in a safe state: neither P1 nor P2 can now be satisfied, the request should not be granted. The process should be placed in the waiting state until a safe state can be entered.**

## Question 2 – Memory Management (10 points total)

### (a) Segmentation with Paging (5 points)

There is one process in a system that currently uses 3 segments that contains the following (a simplified example; the strings represent variable size content, i.e. variable size segments):

Segment	Content
0	<i>Pointers are great but quite a challenge.</i>
1	<i>Data can exist on the stack or heap or data segment.</i>
2	<i>Stack and heap are important.</i>

The "simplified computer" uses an 8-bit address space with paged segments. Each process can have at most 4 segments, each paged with up to 4 pages; the pages contain 16 bytes each. Thus segments can be at most 64 bytes long (4 pages x 16 bytes) with the process at most 256 bytes long (4 segments x 64 bytes) (The content of the physical memory is as follows (ignore the fact that there is no space for the OS itself and other processes):

Frame #	Frame content	Frame meaning
0 (0000)	<i>Pointers are gre</i>	Data/code
1 (0001)	<02, 06, 04, -1>	Segment table
2 (0010)	<00, 08, 09, -1>	Page table of segment 0
3 (0011)	Old_data_here	Empty
4 (0100)	< 13, 15, -1, -1>	Page table of segment 2
5 (0101)	<i>Data can exist o</i>	Data/code
6 (0110)	< 05, 10, 07, 11>	Page table of segment 1
7 (0111)	<i>heap or data segm</i>	Data/code
8 (1000)	<i>at but quite a c</i>	Data/code
9 (1001)	<i>hallenge.</i>	Data/code
10 (1010)	<i>n the stack or h</i>	Data/code
11 (1011)	<i>ent.</i>	Data/code
12 (1100)	xxxx_xxxx_xxxx	Empty
13 (1101)	<i>Stack and heap a</i>	Data/code
14 (1110)	aaaa_bbbb_cccc	Empty
15 (1111)	<i>re important.</i>	Data/code

- (2 points) In the table above, fill in the empty fields in the segment table and in the page tables of the segments. Use -1 to indicate invalid segment/page (only 4 entries exist in each table).
- (1 point) The logical address is 8 bits long as is the physical address. Give the number of bits for the following:
 

Offset in the physical/logical address:	<b>4 bits</b>
Frame number:	<b>4 bits</b>
Segment number:	<b>2 bits</b>
Page number:	<b>2 bits</b>
- (2 points) Fill in the last two columns of the following table:

Logical address	Corresponding physical address (or access violation)	Data/code byte at this address
00000101	<b>0000 0101</b>	<b>e</b>
01110001	<b>1011 0001</b>	<b>n</b>
10111110	<b>Access violation</b>	
01101101	<b>0111 1101</b>	<b>e</b>

The following table shows how 4 physical frames allocated to a process have been loaded with its logical pages (at time 164). All values are numbered using decimal values and starting at 0. All timing values correspond to the number of clock ticks from the start of the process execution. The U bit is the use bit that is set whenever the loaded page is referenced.

Page Frame Number	Virtual Page Number	Time loaded	Time Referenced	U Bit
0	2	60	161	0
1	1	130	160	1
2	0	26	162	1
3	3	20	163	1

For each page replacement algorithm below, circle the virtual page numbers that cause a page fault in the given sequence of virtual page accesses. Under the page number that causes the page fault, show the page replacement by giving the new list of loaded virtual pages (i.e. the contents from the first column of the above table). It is not necessary to show the contents of the list for the virtual pages that do not cause a page fault.

[illegible]

(among the virtual pages 2,1,0,3, 3 is the furthest in the future. At the end, we do not have enough information about the future to decide, we just know that 2 will not be replaced)

Clock      (4)    0    0    0    (2)    4    2    1    0    (3)    2

Assume that a pointer is initially set to Page Frame 2.

0	2	<b>4*</b>	4*	>4*	>4*	>4*	4	4
1	1*	>1*	>1*	1	1*	1*	> <b>3*</b>	>3*
2	>0*	0	0*	0	0	0*	0	0
3	3*	3	3	<b>2*</b>	2*	2*	2	2*

LRU      (4)    0    0    0    2    4    2    (1)    0    (3)    2

0	2	2	2
1	<b>4</b>	4	<b>3</b>
2	0	0	0
3	3	<b>1</b>	1

### Question 3 - Semaphores (5 points)

Using semaphores simulate a unisex washroom with the following rules:

1. Men and women cannot be in the washroom at the same time.
2. At most 3 people can be in the washroom (either 3 men or 3 women).
3. The method `useWashroom()` simulates a person using the washroom and called when the person has the right to enter and use the washroom according to rules 1 and 2.
4. The method `maleUse()` is called by male "threads" to use the washroom; complete **this** method. (The method `femaleUse()` will be used by female "threads" and has the same logic.)
5. In the method, use the following semaphores and variable; give initial values for them in the declarations below.
  - a. Variable `maleNum` to denote the number of males that are allowed to use the washroom (either waiting for it because it is full, or already inside).
  - b. Semaphore `maleAccess` and to control access to the variable (male threads will block on this Semaphore when women are in the washroom).
  - c. Semaphore `maleEnter` and for controlling the number of males entering the washroom (men will block on this Semaphore when 3 men are in the washroom).
  - d. Semaphore `empty` for gaining access to the washroom when the washroom is empty (this semaphore is used by both men and women threads)(Note, the method `femaleUse()` would use the variable `femaleNum`, semaphores `femaleAccess` and `femaleEnter`; and also the semaphore `empty`).
6. Your solution must not deadlock, but do not worry about starvation.

// Semaphores/variable declarations: Show initial values

```
int maleNum = 0
```

```
Semaphore maleAccess=1, maleEnter=3, empty=1
```

```
maleUse()
{
    maleAccess.wait()
    maleNum++
    if(maleNum == 1) empty.wait();
    maleAccess.signal();
    maleEnter.wait(); // only 3 at a time

    useWashroom();

    maleEnter.signal();
    maleAccess.wait()
    numMale--
    if(numMale == 1) empty.signal();
    maleAccess.signal();
}
```

### Question 5 - File Systems (10 points)

The exam annex provides a description of the contents of the Quantum Mechanics<sup>1</sup> Minix File System including:

- A diagram that illustrates the organization of the directories and files in the Minix file system; directory names are in bold font and file names in regular font.
- A block allocation table that gives the block numbers (Minix zone numbers) that indicate where the content of the directories and files are located; the order of the physical block numbers reflect the order of the file/directory logical blocks.
- The listing produced by “ls -lR” executed from the root of the mounted Minix file system.

Using the available information in the annex, complete the Minix file system structures on the following pages: the inodes, directory tables and index blocks. It is NOT necessary to fill in the fields that are grayed out.

---

<sup>1</sup> The directory names are names of scientists who contributed to the development of quantum mechanics and the file names are related to subjects they studied and developed.

# MINIX File System

MINIX Inode Table							
inode #	i_mode	i_uid	i_size	i_time	i_gid	i_nlinks	i_zone[0..8]
1	0100 000 111 101 101		128			2	524
2	0100 000 111 101 101		224			5	525
3	0100 000 111 101 101		128			2	555
4	0100 000 111 101 101		192			3	543
5	0100 000 111 101 101		128			2	549
6	0100 000 111 101 101		128			2	526
7	0100 000 111 101 101		128			2	587
8	1000 000 110 100 100		2496			1	564, 565, 566
9	1000 000 110 100 100		5120			1	556, 557, 558, 559, 560
10	1000 000 110 100 100		5120			1	544, 545, 546, 547, 548
11	1000 000 110 100 100		2496			4	561, 562, 563
12	1000 000 110 100 100		16640			1	569, 570, 595, 596, 597, 598, 599, 600
13	1000 000 110 100 100		448			1	530
14	1000 000 110 100 100		448			1	583
15	1000 000 110 100 100		5120			1	567, 568, 577, 578, 579
16	1000 000 110 100 100		1664			1	614, 615
17	1000 000 110 100 100		5120			1	588, 610, 611, 612, 613
18							
19							
20							
21							
22							
23							
24							
25							



## MINIX Directory Tables

<b>Directory: <i>root</i></b>	
<b>Ino</b>	<b>Name</b>
<b>1</b>	<b>.</b>
<b>1</b>	<b>..</b>
<b>3</b>	<b>bohr</b>
<b>2</b>	<b>plank</b>

<b>Directory: bohr</b>	
<b>Ino</b>	<b>Name</b>
<b>3</b>	<b>.</b>
<b>1</b>	<b>..</b>
<b>8</b>	<b>atom</b>
<b>9</b>	<b>hydrogen</b>

<b>Directory:plank</b>	
<b>Ino</b>	<b>Name</b>
<b>2</b>	<b>.</b>
<b>1</b>	<b>..</b>
<b>10</b>	<b>blackbody</b>
<b>11</b>	<b>constant</b>
<b>5</b>	<b>deBroglie</b>
<b>6</b>	<b>einstein</b>
<b>4</b>	<b>schrodinger</b>

<b>Directory: deBroglie</b>	
<b>Ino</b>	<b>Name</b>
<b>5</b>	<b>.</b>
<b>2</b>	<b>..</b>
<b>13</b>	<b>duality</b>
<b>11</b>	<b>planksConstant</b>

<b>Directory:einstein</b>	
<b>Ino</b>	<b>Name</b>
<b>6</b>	<b>.</b>
<b>2</b>	<b>..</b>
<b>12</b>	<b>photon</b>
<b>11</b>	<b>plConstant</b>

<b>Directory: schrodinger</b>	
<b>Ino</b>	<b>Name</b>
<b>4</b>	<b>.</b>
<b>2</b>	<b>..</b>
<b>7</b>	<b>born</b>
<b>14</b>	<b>cat</b>
<b>15</b>	<b>equation</b>
<b>11</b>	<b>plankC</b>

<b>Directory: born</b>	
<b>Ino</b>	<b>Name</b>
<b>7</b>	<b>.</b>
<b>4</b>	<b>..</b>
<b>17</b>	<b>mechanics</b>
<b>16</b>	<b>quantum</b>

The following table can be used to define index blocks. You may use any data block (i.e. block number) not used in Table A1 of the annex (i.e. block allocated to content of directories and files. Data blocks start at block number 524.

<b>Index Blocks</b>	
<b>Index Block Number</b>	<b>Index Block Content</b>
600	601, 602, 603, 604, 605, 606, 621, 622, 623