IE OS ASSIGNMENT-2

1. (I am writing the code for this question over core as i couldn't execute the code as there was some error with file location)

Code for hello.c:

```
#include<linux/init.h>
#include<linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
	printk(KERN_ALERT "Hello world.\n");
	return 0;
}

static void hello_exit(void)
{
	printk(KERN_ALERT "Goodbye world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Code for makefile:

```
obj-m += hello-1.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

2. Consider the processes P1, P2, P3, P4 whose arrival times are 0, 2, 3, 5 and burst times are 7,4, 2, 4 respectively. What is the average TAT if they follow the Shortest Remaining Time First scheduling algorithm?

Answer: The various Programs will get executed as follows:

P1 => 0-2; P2=> 2-3; P3=>3-5; P2 => 5-8; P4 => 8-12; P1 => 12-17.

Therefore the Turnaround time for each process will be as follows:

P1 = 17-0 = 17;

P2 = 8-2 = 6;

P3 = 5-3 = 2;

P4 = 12-5 = 7

Therefore, average TAT = $(17 + 6 + 2 + 7) \div 4 = 8$

3. Given reference to the following page by a program
0,9,0,1,8,1,8,7,8,7,1,2,8,2,7,8,2,3,8,3
If the program contains 3 page frames. How many page faults will occur in optimal page replacement policy?
Answer: In optimal page replacement policy, the pages that would not be used for the longest duration of time in the future will be replaced.

0	9	0	1	8	1	8	7	8	7	1	2	8	2	7	8	2	3	8	3
0	0	X	0	9	X	X	1	X	X	X	8	X	X	X	X	X	8	X	X
	9		9	1			8				7						2		
			1	8			7				2						3		

- Since all 3 slots are initially empty, 3 page faults will occur. => 3 page faults
- When 8 comes, it will replace 0 as it will not be used for the longest time in the future. => 1 page fault.
- At 1 and 8, no page faults will occur as they are already present. => 0 page faults
- When 7 comes, it will replace 9 as it will not be used for the longest time in the future. => 1 page fault.
- No page faults will occur at 8, 7 and 1 as they are already present => 0 page faults
- When 2 comes, it will replace 1 as it will not be used for the longest time in the future. => 1 page fault.
- No page faults will occur at 8, 2, 7, 8 and 2 as they are already present => 0 page faults
- When 3 comes, it will replace 7 as it will not be used for the longest time in the future. => 1 page fault.
- No page faults will occur at 8 and 3 as they are already present => 0 page faults

Therefore , the total number of page faults that will occur is 7.

4. Consider the virtual page reference string 1, 2, 3, 2, 4, 1, 3, 2, 4
On a demand paged virtual memory system running on a computer system that has a main memory size of 3 page frames which are initially empty. Then calculate the number of page faults in FIFO and LRU respectively.
Answer: For FIFO,

1	2	3	2	4	1	3	2	4
1	1	1	X	2	3	X	4	X
	2	2		3	4		1	

		3		4	1		2	

Therefore, total number of page faults = 6.

For LRU,

1	2	3	2	4	1	3	2	4
1	1	1	X	4	4	4	2	2
	2	2		2	2	3	3	3
		3		3	1	1	1	4

Therefore, total number of page faults = 8.

5. Consider three processes P1, P2 and P3 arrive at the same time zero. The burst time of the processes is 3, 3 and 24 respectively. Consider the first come first serve (FCFS) scheduling algorithm. Then what is the throughput (Number of processes completed per unit time)?
Answer: According to FCFS, the processes will be executed as follows:
P1 => 0-3; P2=> 3-6; P3 => 6-30.
Total time for execution of all processes = 3+3+24 = 30 and
Total number of processes = 3
Therefore, throughput = 3/30 = 0.1 processes per unit time

6. The following C program is executed on a Linux/Unix system:

```
int main()
{
int  i ;
for (i=0; i<10; i++)
if(i%3==0)  fork();
return 0;
}
```

The total number of processes created is(state your reasons as well):

Answer: The above program will call fork() 4 times i.e. for the values of i=0,3,6,9.
The total number of processes created = $2^n - 1$,
Where n is the number of times fork() is called.
Therefore, in this case, total number of processes created = $2^4 - 1 = 15$

7.Name two page replacement Algorithms that do not suffer from Belady's anomaly.

Answer: The two page replacement Algorithms that do not suffer from Belady's anomaly,

- Optimal Page replacement Algorithms
- Least Recently Used (LRU) Page replacement Algorithm.

8. Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300KB, and 250 KB, where KB refers to kilobyte. These partitions need to be allotted to four processes of sizes 190 KB, 210 KB, 368 KB and 391 KB in that order. If the best fit algorithm is used, which partitions are NOT allotted to any process?
Answer: The best fir algorithm leads to the following partitions,
P1 190KB => 200KB
P2 210KB => 250KB
P3 368KB => 400KB
P4 391KB => 500KB
Therefore, the partitions that will not be allotted to any process is 300KB and 600KB.

9. Which Structure is needed for converting logical address to physical address?
Answer: Table Structure specifies the translation between Virtual address and Physical Address. The hardware device Memory Management Unit (MMU) does the run time mapping between Virtual address and Physical Address.

10. Adding more frames can cause more page faults is known as ??
Answer: Belady's Anomaly is the name given to the phenomenon where adding more frames can cause more page faults for a given memory access pattern.

11. Calculate the size of memory if its address consists of 24 bits and the memory is 8-byte addressable.
Answer: Given that the address consists of 24 bits. Therefore,
Number of locations possible with 24 bits = 2^{24}
Also, Size of one location = 8 bytes
We have, Size of memory = Number of locations * size of one location
Therefore, Size of memory = $2^{24} \times 8$ bytes = 2^{27} bytes
= 134 MB

12.Consider a machine with 64 MB physical memory and a 22 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?
Answer: Size of physical memory = 64 MB
Number of bits in virtual address space = 22 bits
Page Size = 4KB
We have, 64MB = 2^{26} Bytes
Therefore, total number of bits in physical address = 26.

Number of frames in main memory = (Size of main memory) ÷ (Size of each frame)

= 64MB ÷ 4KB = $2^{26} \div 2^{12} = 2^{14}$

Therefore, number of bits in frame number = 14

Also, Page size = 4KB = 2^{12} bytes.

Therefore, number of bits in page offset = 12 bits

Number of bits in virtual address space = 22 bits.

Therefore, Process size = 2^{22} Bytes = 4 MB

Number of pages the process is divided into = Process size ÷ Page size

= 4MB / 4KB = 2^{10} pages

Thus, Number of entries in page table = 2^{10} entries

Therefore, Page table size = Number of entries in page table x Page table entry size

= Number of entries in page table x Number of bits in frame number

= $2^{10} \times 14$ bits

= $2^{10} \times 16$ bits (Approximating $16 \approx 14$ bits)

= 2^{14} bits = 16KB

13.Assuming 1KB page size what are page number and offset for following address references :(give page number and page offset) 1) 2375 2) 30000

Answer: For a 1KB(1024 bytes) page size,

Page number = address/1024 and

Page offset = address % 1024

i) For address reference 2375,

Page number = $2375 \div 1024 = 2$ and

Page offset = 2375 % 1024 = 327

ii) For address reference 30000,

Page number = $30000 \div 1024 = 29$ and

Page offset = 30000 % 1024 = 304

14.Suppose We want to Sync processes P and Q using semaphores's S and T, process P and Q are defined as follows:

You need to sync such that patterns 011..1110 or 100..0001 should not get printed. This has repeating one and zeros in the middle. The number of times it gets repeated is odd. Here which of the following values of w,x,y,z is possible??

Optins: Signal =s() Wait=w()

A) w(S), s(S), w(T), s(T)

B) w(S), s(T), w(T), s(S)

C) w(S), s(S), w(S), s(S)

Answer: C)

To support mutual exclusion in process ‘P’ and ‘Q’, we have S=1 and T=0 initially.

The wait function and signal function are defined as follows,

```
Wait(S)  {
   while (i <= 0) ;
   S-- ;
}
Signal(S)  {
   S++ ;
}
```

Since we have S=1, Process ‘P’ gets executed and w(S) will decrement the value of S by one to make S=0. Simultaneously, in process Q, T=0 and hence, process ‘Q’ does not do anything until process ‘P’ prints “00” and increments the value of T by calling the function s(T).

Now, while the control is in process ‘Q’, we have S=0 and “11” gets printed. Process ‘P’ does not execute anything until “11” is printed since S=0. After printing “11”, process ‘Q’ calls the function w(S) and increments the value of S to S=1.

In this way, both the process ‘P’ and ‘Q’ coordinate to give the output “001100…” and not patterns like 011..1110 or 100..0001.