


National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Operating Systems	Course Code:	CS 205
	Program:	Bachelors in Computer Science	Semester:	Spring 2019
	Duration:	180 minutes	Total Marks:	55
	Paper Date:	14 th May, 2019	Weight	45
	Section:	ALL	Page(s):	9
	Exam Type:	Final		

Student : Name: _____ Roll No. _____ Section: _____

Instruction/Notes: Attempt all questions. Programmable calculators are not allowed.

Question 1: [12 marks]

Please write your answer in the table given below. Answers outside the table will not be marked.

Question no.	Your answer
1.1	a
1.2	b
1.3	b
1.4	F
1.5	T
1.6	F
1.7	F
1.8	a,b
1.9	c,d
1.10	a
1.11	c
1.12	a

1.1 Factors effecting the context-switch time are

- a. Register sets available at the hardware layer
- b. Number of processes in Ready Queue
- c. Format of FCB
- d. Size of physical memory

1.2 Consider calling fork() system call on Unix based operating system, which of the followings will be true after successful execution of fork

- a. Instruction following the fork() system call will be executed in the parent process before the child process gets CPU
- b. Parent and child processes will be running concurrently
- c. Parent and child processes share the data, hence modification to a variable will be visible to both processes
- d. Parent process cannot terminate until all the child processes terminate

1.3 A parent process terminates without calling wait() for its child processes. What possible scenarios can happen?

- a. Zombie Processes count increases
- b. Creation of Orphan processes
- c. Termination of all child processes
- d. Init process invoked to clear all zombie and orphan process

1.4 The OS provides the illusion to each thread that it has its own address space. True/False?

1.5 Threads belonging to the same process can access the same TLB entries. True/false?

1.6 Peterson's algorithm uses the atomic instructions to provide mutual exclusion for two threads. True/False?

1.7 File update operation is considered to be slow because multiple copies of the same FCB are maintained in memory and it takes time to maintain consistency among these duplicates. True/false?

1.8 Which of the following structure(s) must be stored with a file system persistently

- a. Boot Control block
- b. FCBs
- c. System-wide open file table
- d. Mount table

1.9 Which of the following structures are maintained in memory for proper file system execution

- a. Directory Structure
- b. Volume Control Block
- c. Mount table
- d. Per process open-file table

1.10 Which of the following allocation methods can cause external fragmentation

- a. Contiguous Allocation
- b. Linked Allocation
- c. Indexed Allocation
- d. None of the above

1.11 Consider the code snippet given below (assume irrelevant details have been omitted).

```
int balance = 0; //this is a global variable
void *mythread(void *arg) {
    inti;
    for (i = 0; i < 200; i++) {
        balance++;
    }
    printf("Balance is %d\n", balance);
    return NULL;
}
int main(intargc, char *argv[])
    pthread_t p1, p2, p3;
    pthread_create(&p1, NULL, mythread, "A");
    pthread_join(p1, NULL);
    pthread_create(&p2, NULL, mythread, "B");
    pthread_join(p2, NULL);
    pthread_create(&p3, NULL, mythread, "C");
    pthread_join(p3, NULL);
    printf("Final Balance is %d\n", balance);
}
```

What is the output of the code when thread p2 prints "Balance is %d\n"

- a. Due to race conditions, "balance" may have different values on different runs of the program.
- b. 200
- c. 400
- d. 600
- e. None of the above

1.12 Consider the code snippet given below (assume irrelevant details have been omitted).

```
int balance = 0; //this is a global variable
void *mythread(void *arg) {
    inti;
    for (i = 0; i < 200; i++) {
        balance++;
    }
    printf("Balance is %d\n", balance);
    return NULL;
}
int main(intargc, char *argv[])
```

```
pthread_t p1, p2, p3;
pthread_create(&p1, NULL, mythread, "A");
pthread_create(&p2, NULL, mythread, "B");
pthread_create(&p3, NULL, mythread, "C");
pthread_join(p1, NULL);
pthread_join(p2, NULL);
pthread_join(p3, NULL);
printf("Final Balance is %d\n", balance);
}
```

What is the output of the thread p3 when it executes the statement "Balance is %d\n"

- a. Due to race conditions, "balance" may have different values on different runs of the program.
- b. 200
- c. 400
- d. 600
- e. None of the above

Question 2: [4 marks]

Consider the following code snippet that creates multiple processes. Assume that the sleep(x) method puts the process in the waiting queue for x seconds.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid;

    pid = fork();
    if (pid == 0) { /* Lets call this process 1 */
        sleep(5);
    }
    else {
        pid_t pid1;
        pid1 = fork();
        if (pid1 == 0)
        { /* Lets call this process 2 */
            Sleep(20);
        }
        else
        {
            sleep(10);
            Wait(NULL);
        }
        return 0;
    }
}
```

- a. Which process(es) becomes the Zombie process? P1
For how much time? 5 secs
- b. Which process(es) becomes the Orphan process? P2
For how much time? Until Init process calls the wait

Question 3: [6 marks]

Consider the following set of processes, with the length of the CPU burst, initial priority and arrival time. A larger priority number implies a higher priority. I/O burst are not required by these processes.

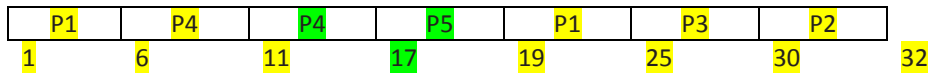
	CPU Burst	Initial Priority	Arrival time
P1	11	2	1
P2	2	1	2
P3	5	2	3
P4	11	3	4
P5	2	3	5

Draw a **Gantt Chart** for the above mentioned processes using the following scheduling algorithm.

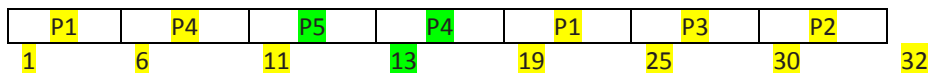
"Scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 5. However, every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 1 is added to its time quantum, and its priority level is incremented. When a process blocks before using its entire time quantum, its time quantum is reduced by 1, but its priority remains the same."

Use the following table to show your gantt chart. Describe the running process inside the cell while mentioning the scheduling time below it as shown for the P1.

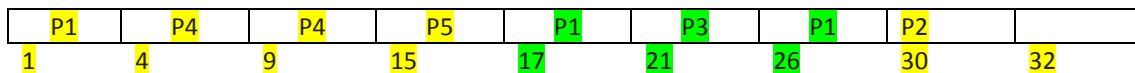
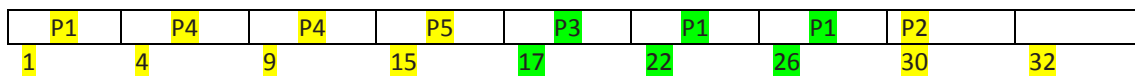
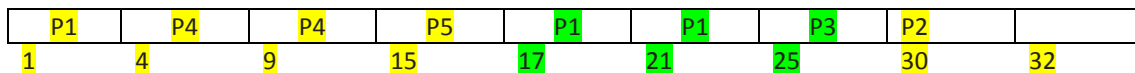
Option 1: Non-Pre-emptive Scheduler (100 % correct solution – deduct 1 mark for using non-pre-emptive while the question asked for pre-emptive solution)



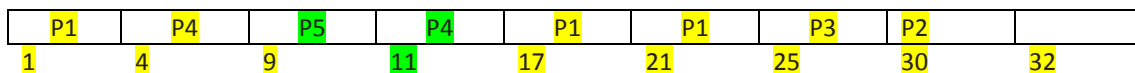
Option 2: Non-Pre-emptive Scheduler (P5 scheduled before P4 – deduct 1 or 1.5 marks)



Option 3: Pre-emptive Scheduler (100% correct solution – 3 variants)



Option 4: Pre-emptive Scheduler (P5 scheduled before P4 – deduct 1 to 1.5 marks)



Rough Work:

Question 4: [6 marks]

A tribe of savages eats communal dinners from a large pot that can hold M servings of stewed missionary¹. When a savage wants to eat, he helps himself from the pot, unless it is empty. If the pot is empty, the savage wakes up the cook and then waits until the cook has refilled the pot.

Any number of **savage threads** run the following code:

```
while(true) {
    getServingFromPot();
    eat (); }
```

And one cook thread runs this code:

```
while(true){
    putServingsInPot(M); }
```

The synchronization constraints are:

- Savages cannot invoke `getServingFromPot` if the pot is empty.
- The cook can invoke `putServingsInPot` only if the pot is empty.

Puzzle: Add code for the savages and the cook that satisfies the synchronization constraints.

Hint: you need to use the following variables to complete the write-up of this puzzle:

- `Servings = 0;` //variable to hold the number of servings left
- `Mutex = Semaphore(1);` // you know what this is for
- `emptyPot = Semaphore(0);` //indicates that the pot is empty
- `fullPot = Semaphore(0);` //indicates that the pot is full

Your code goes below:

Cook Thread:	Savages Thread:
<pre>while(True){ putServingsInPot(M); }</pre>	<pre>While (true){ GetServingFromPot(); Eat(); }</pre>

Dining Savages solution (cook)

```
1 while True:
2     emptyPot.wait()
3     putServingsInPot(M)
4     fullPot.signal()
```

Dining Savages solution (savage)

```
1 while True:
2     mutex.wait()
3     if servings == 0:
4         emptyPot.signal()
5         fullPot.wait()
6         servings = M
7         servings -= 1
8         getServingsFromPot()
9     mutex.signal()
10
11 eat()
```

Question 5: [6 marks]

Consider the following code for a simple Stack:

```
class Stack {
private:
    int* a; // array for stack
    int max; // max size of array
    int top; // stack top

    Semaphore full = 0;
    Semaphore empty = MAX;
    Semaphore mutex = 1;

public:
    Stack(int m) {
        a = new int[m];
        max = m;
        top = 0;
    }
    void push(int x) {
        wait(empty);
        wait(mutex);
        a[top] = x;
        ++top;
        signal(mutex);
        signal(full);
    }
    int pop() {
        wait(full);
```

```

        wait(mutex);
        int tmp = top;
        --top;
        signal(mutex);
        signal(empty);
        return a[tmp];
    }
};

```

You can see from the code that a process does busy waiting if it calls push() when the stack is full, or it calls pop() when the stack is empty.

Consider running the functions push and pop concurrently, synchronize the code using semaphores with the following requirements

- Eliminate the busy waiting
- Push should not be allowed to put item into stack if the stack is full.
- Pop should not be allowed to pop an item if the stack is empty

Just modify the code clearly in the box above. No need to rewrite.

Question 6: [8 marks]

Assume an architecture where every virtual address is of 16-bits and it is translated to a physical address using a 2-level paging hierarchy. Each virtual address has the format:

3-bits	3-bits	10-bits
Page Dir(outer page table) offset	Page Table (inner page table) Offset	Page Offset

Each page-table (inner page-table) entry and page-directory (outer page table) entry is of 12-bits. Both have the same format given below:

6-bits	3-bits	1-bit	1-bit	1-bit
Base address of Page-Table/Page	Reserved	Read	Write	Valid

Read and Write bits, when set, indicate that reading and writing is enabled respectively. If the valid bit is set, it indicates that the page is in physical memory otherwise it is not present. Three bits are reserved and are ignored for all practical purposes.

<i>Note: Give answers in bytes, KB, or MB when asked about memory size.</i>	
a. What is the size of a single page table? (1 mark)	12 bits x 2 ³ = 12 bytes.
b. What is the size of a single physical page? (1 mark)	2 ¹⁰ = 1 KB
c. What is the maximum amount of virtual memory available to any process? (1 mark)	2 ¹⁶ = 64 KB
d. What is the size of a physical address in number of bits?(1 mark)	16 bits
e. What is the maximum physical memory that can be utilized by a user process in this system? (1 mark)	64 KB
Using the same architecture, you are given the following state of page directory (outer page table) and page tables (inner page table) for a particular process as shown in Figure 1. . All values are in hex. Addresses increase from bottom to top of each table.	
f. What is the total amount of physical memory currently being used by the process?(3 mark)	12 KB

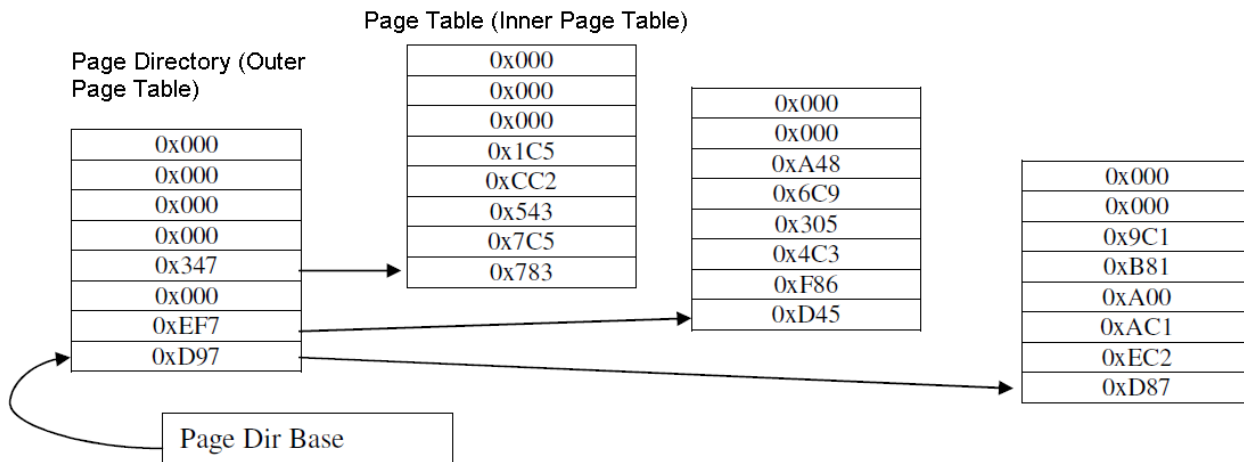


Figure 1: Page Directory (Outer Page Table) and Page Table (Inner Page Table) State for a Process

Question 7: [7 marks]

Consider the following code:

```
constint size = 4194304;           //4 MB
char* buf = (char*) malloc(size);  //Allocate 4MB virtual memory
char* end = buf + size;
char* p;
for(p = buf; p < end; p += 8192)
    *p = random();                //Write a random value into the location pointed-to by p
```

The malloc() library call allocates a 4 MB chunk of virtual memory from the heap portion of a process and returns the virtual address of the first byte (which is stored in **buf**). Assume that NONE of this 4MB virtual memory is mapped to physical memory (i.e. as the virtual pages are accessed they are mapped to physical memory and page table entries are setup). Also assume that the OS and hardware are using paging with a page size of 4KB. Assume that you have enough physical memory, so that you never need to replace a page. Please note that a char consists of one byte.

a. Will the OS be able to run this code without actually using 4MB of physical memory for buf? Answer “yes” or “no”. (1 mark)	Yes	
b. How much physical memory (for buf array) will actually be in use once this loop ends? Provide answer in MB’s. (1 mark)	512 * 4 KB = 2 MB	
c. How many page faults does the “for loop” cause?(1 mark)	512 page faults	
d. Assume now that the page size in your hardware is 8KB instead of 4KB. Write the total page faults and total physical memory used (in MBs) in the right box. (2 marks)	Total Page Faults	512
	Total Physical Memory Usage (in MB’s)	4MB
e. Assume that we change the “for loop” in the above code to the following. Fill the table in the right. (2 marks) for(p = buf + 2048; p < end; p += 4096) *p = random();	Total Page Faults	
	Page-size = 4 KB	1024
	Page-size = 8 KB	512

Question 8: [6 marks]

Consider a file system on a disk that has both logical and physical block sizes of 1KB(1024 bytes). Assume that the FCBs of each file are already in memory, however, every other relevant data structure must be read from the disk.

a. If we are currently at logical block 8 and want to access logical block 4, how many physical blocks must be read from the disk for each of the following methods?

1. Contiguous Allocation Method

Answer: 1

2. Linked Allocation method

Answer: 4

3. Indexed Allocation method

Answer: 1

b. Consider a file system that used Combined Indexed Allocation method with following details

1. Index table contains 12 direct block addresses
2. Index table contains single, double and triple indirect block addresses
3. Disk blocks are 1 KB in size
4. Pointer to a disk block requires 4 bytes

What is the maximum size of a file that can be stored in this file system? Show your complete calculations below. [Answer should be in Kilobyte units. No need for unit conversions].

Maximum file Size : _____

Calculations:

w = Data stored through direct block addresses: $12 * 1\text{KB}$

1 block can contain block addresses = $1\text{KB} / 4\text{B} = 2^8 = 256$ addresses

x = Data stored through single indirect block addresses:

$256 * 1\text{KB}$

y = Data stored through double indirect block addresses:

$256 * 256 * 1\text{KB}$

z = Data stored through triple indirect block addresses:

$256 * 256 * 256 * 1\text{KB}$

Maximum File size = $w + x + y + z$