National University of Computer and Emerging Sciences, Lahore Campus

| | | | | |
|---|---|---|---|---|
| | Course: | Operating System | Course Code: | CS-205 |
| | Program: | BS(Computer Science) | Semester: | Fall 2017 |
| | Duration: | 1 hour | Total Marks: | 50 |
| | Paper Date: | 2$^{nd}$ November, 2017 | Weight: | 15% |
| | Section: | All | Page(s): | 3 |
| | Exam: | Mid-2 | Roll No. | |

**Instructions/Notes:** Answer questions on the question paper. Write answers clearly and precisely, if the answers are not easily readable then it will result in deduction of marks. Use **extra sheet** for rough work, cutting and blotting on this sheet will result in deduction of marks.

**Question 1 (10 points):** Given below are two tables. The first table shows the processes coming **in order** (a process with lower number comes earlier than the one with greater number). Using **best fit** algorithm assign each process some hole given in the following table. Also calculate the remaining space.

| Process# | Required Memory (MBs) |
|---|---|
| 1 | 20 |
| 2 | 43 |
| 3 | 21 |
| 4 | 77 |
| 5 | 6 |
| 6 | 11 |
| 7 | 15 |
| 8 | 105 |
| 9 | 62 |
| 10 | 90 |

| Hole# | Hole Size (MBs) | Allotted Process# | Remaining space |
|---|---|---|---|
| 1 | 83 | 4,5 | 6,0 |
| 2 | 50 | 2 | 7 |
| 3 | 100 | 10 | 10 |
| 4 | 25 | 1 | 5 |
| 5 | 30 | 3 | 9 |
| 6 | 10 | | |
| 7 | 70 | 9 | 8 |
| 8 | 15 | 6 | 4 |
| 9 | 17 | 7 | 2 |
| 10 | 110 | 8 | 5 |

**Question 2 (4 points):** If you use **worst fit** algorithm instead, then which hole will be assigned to first **two** processes

- Process 1 → Hole#= 10
- Process 2 → Hole#= 3

**Question 3 (6 points):** Extract page numbers and offsets from the following logical addresses. The page size is 1000 bytes.

- 2101 → Page#=2 Offset=101
- 5215 → Page#=5 Offset=215
- 102 → Page#=0 Offset=102

**Question 4 (15 points):** Using the below given functions you have to implement a function `handlePageFault`. It is called when a page fault occurs. Meaning, when a page is not found loaded into the memory, it load it and fixes the page table. Use following functions without the knowledge of their definition. Comments describe their functionality. The page table is a **two level page table** stored without caching. **Hint:** read the declarations carefully!

```
int getFirstLevelPageNumber(int logicalAddress); // takes the faulty logical address as input
    and returns the associated first level page number.

int getSecondLevelPageNumber(int logicalAddress); // takes the faulty logical address as input
     and returns the associated second level page number.

int getCombinedPageNumber(int logicalAddress); // takes the faulty logical address as input
    and returns the value of bits associated to the page number (first and second level).

int* getFirstLevelPageTable(int PID); // takes the process ID as input and returns the pointer
     to its first level page table.

int loadPageFromBackingStore(int combinedPageNumber, int PID); // takes the page number and
    process ID as inputs and returns the frame number on which it loaded the page.
```

```
void handlePageFault( int logicalAddress, int PID)//the faulty address and the process ID are
    the parameters to the function.
{

        int firstP = getFirstLevelPageNumber(logicalAddress);
        int secondP = getSecondLevelPageNumber(logicalAddress);
        int combinedP = getCombinedPageNumber(logicalAddress);

        int* firstPT = getFirstLevelPageTable(PID);
        int* secondPT = firstPT[firstP];

        int frame = loadPageFromBackingStore(combinedP, PID);
        secondPT[secondP] = frame;

}
```

**Question 5 (15 points):** Following is a proposed solution for readers writers problem. Although the solution fulfills some requirements, but it does violate "The Bounded Wait" property. Which means that in this case the writer may have to wait indefinitely. Remove that problem by inserting new semaphore(s). For 100% marks propose a solution which gives a better chance to writers to enter into the critical section.

| Code for Writers | Code for Readers |
|---|---|
| roomEmpty=0,mutex=0, readers=0 // all variables are shared | |

Code for Writers:

```
$roomEmpty.wait()$
    Write($Rsc$)
$roomEmpty.post()$
```

Code for Readers:

```
$mutex.wait()$
    $readers++$
    if $readers == 1$ :
        $roomEmpty.wait()$
$mutex.post()$
Read($Rsc$)
$mutex.wait()$
    $readers--$
    if $readers == 0$ :
        $roomEmpty.post()$
$mutex.post()$
```

| Code for Writers | Code for Readers |
|---|---|
| roomEmpty=0,mutex=0, readers=0 sem = 0// all variables are shared | |

Code for Writers:

```
$sem.wait()$
$roomEmpty.wait()$
    Write($Rsc$)
$sem.post()$
$roomEmpty.post()$
```

Code for Readers:

```
$sem.wait()$
$sem.post()$
$mutex.wait()$
    $readers++$
    if $readers == 1$ :
        $roomEmpty.wait()$
$mutex.post()$
Read($Rsc$)
$mutex.wait()$
    $readers--$
    if $readers == 0$ :
        $roomEmpty.post()$
$mutex.post()$
```