Sample Midterm Answers

First solve without referring to answers.

1a) Refer to lecture-5 slides

1b) A Heap can be shared among threads and is desirable. While threads get their own stack, threads can share stacks, although not desirable. While the general practice is to avoid stack sharing and use thread's private stack, even when it needs to be done, access must be synchronized. Below is an example of stack sharing and problems.

Example

Preparation

```
int *p;
phread_create(..., thread #1, ...);
phread_create(..., thread #2, ...);
```

Thread #1

```
int q[1024]; // Allocated on Thread #1's stack
p = q;
q[0] = 1;
```

Thread #2

```
*p = 2; // Thread #1's stack accessed
```

If thread #1 and thread #2 are concurrent, and there is no other synchronization between them, resulting in cross-thread stack access. A problem could occur if "p = q;" in thread #1 executes before "*p = 2;" executes in thread #2.
The example shows the case where thread #1 publishes the address of a local stack variable q to the global pointer p, and later thread #2 accesses the stack of thread #1 by dereferencing the global pointer p, in effect writing to the variable q on thread #1.

Possible Correction Strategies

- Keep stack variables private to each thread.
- Avoid publishing addresses of stack variables and storing shared data on thread's stack. Instead, store shared data on the heap or in static variables.

1c) The answer for 1b would help you answer this.

2a) See lecture slides "lec2-3-cpuvirt.pdf"

2b and 2c) Please see lecture slides, read the book, or even search the web to find this out as this

is a basic question. If you cannot find it out, send us an email.

3a) We would use 19 bits for the virtual page number and 13 bits for the offset within a page (8K page), assuming a single-level linear page table. So, there will be 512K table entries for a total of 2 MB of physical memory for the page table.

3b) An inverted page table consumes much lesser memory than multiple multi-level page tables as there is only one page-table shared across processes.
However, inverted page tables are difficult to use (Chapter 20). Page sharing is easier with multi-level page tables than with inverted page tables.

3c) Small page sizes reduce the risk of internal fragmentation and enable shorter disk transfers. Large page sizes reduce the size of the page table and increase the sequentiality of disk accesses.

4a) the code contains an atomicity problem because the test of thd->proc_info and its use in the "fputs" statement are not in a critical region. After the test, the value of thd- >proc_info could be changed to NULL by Thread 2

5a) Since virtual address spaces are vast (e.g., 2^64 bytes for a 64-byte address space), many entries for virtual pages in a flat page table are unallocated. Instead of wasting space for them, a multi-level page table can eliminate entire chunks of the address space by storing NULL at different levels.

5b) Level 1 (L1) covers 4KB, L2 covers 512 x 4KB or 2MB, L3 covers 512 x 2MB or 1GB, and L4 covers 512 x 1GB or 512 GB.