Operating Systems – 2011-2012

1 a) User-level threads

Advantages

- Better performance
  - Thread creation and termination is fast
  - Thread switching is fast.
  - Thread synchronisation is fast
  - All these operations do not require any kernel activity
- Allows application specific run-times
  - Each application can have its own scheduling algorithm

Disadvantages

- Blocking system calls stop all threads in the process
  - Denies one of the core motivations for using threads.
- Non-blocking I/O can be used (e.g. select())
  - Harder to use and understand, inelegant
- During a page fault the OS blocks the whole process
  - But other threads can be runnable.
- Difficult to implement preemptive scheduling
  - Runtime can request a clock interrupt but its messy to program.

Kernel-level Threads

Advantages

- Blocking system calls / page faults can be easily accomodated
  - If one thread calls a blocking system call or causes a page fault, the kernel can schedule a runnable thread from the same process

Disadvantages

- Thread creation and termination are more expensive
  - Requires kernel call
  - But still much cheaper than process creation / termination
  - One mitigation strategy is to recycle threads (thread pools)
- Thread synchronization more expensive.
  - requires blocking system calls.
- Thread switching is more expensive
  - Requires kernel call
  - But still much cheaper than process switches
  - Same address space.
- No application specific scheduler.

b i)

int y = x (1)	int z = x (2)	int y = x (1)	int y = x (1)	int z = x (2)	int z = x (2)
x = y + 1 (1)	x = z - 1 (2)	int z = x (2)	int z = x (2)	int y = x (1)	int y = x (1)
int z = x (2)	int y = x (1)	x = y + 1 (1)	x = z - 1 (2)	x = y + 1 (1)	x = z - 1 (2)
x = z - 1 (2)	x = y + 1 (1)	x = z - 1 (2)	x = y + 1 (1)	x = z - 1 (2)	x = y + 1 (1)
0	0	-1	1	-1	1

6 possible combinations of thread interleaving as seen here.

x can be 0, 1, -1 with equal probability $\frac{1}{3}$

ii) Semaphore ~~are~~ is initialised to 1

```
sema.down()          sema.down()
int y = x            int z = x
x = y + 1            x = z - 1
sema.up()            sema.up()
```

iii) Two semaphores initialised to 0

```
int y = x            int z = x
s1.down()            s1.up()
s2.up()              s2.down()
x = y + 1            x = z - 1
```

c i) In virtualisation, all instructions are carried out in user mode, hence those instructions which should be executed in kernel mode are unable to do so, which could mean that the execution of the instruction ~~will~~ will cause undetermined behaviour, for example crashing the system.

ii) Paravirtualisation is when sensitive instructions are replaced by hypervisor calls in the source code of the guest operating system which avoids the context switch overhead of Type 2 hypervisors, thus leading to its better performance. In a Type 2 hypervisor sensitive instructions are caught and emulated (which is called binary translation). The trap and emulate approach has significant overhead due to context switching.

iii) The double-paging problem occurs in the following situation:
Suppose the meta-level policy (i.e. hypervisor) selects a page to reclaim and pages it out. If the guest OS is under memory pressure, it may choose the very same page to write to its own virtual paging device. This will cause the page contents to be faulted in from the system paging device, only to be immediately written out to the VM virtual paging device.
The DPP occurs because the information to avoid this is not available to the hypervisor (parent & guest system demands are not known to each other).

The Ballooning technique allows the guest OS to intelligently make hard decisions about which pages to be paged out without the hypervisor involvement.

2 a) A page table is a data structure used by a virtual memory system in an OS to store the mapping between virtual and physical address. Virtual addresses are used by the accessing process, while physical addresses are used by the hardware.

b) A page fault can occur in the following situations:
1) ~~[illegible]~~ An invalid page fault occurs when an invalid virtual addresses is referenced, which means that there cannot be a page in memory corresponding to it.
2) A minor (soft) page fault occurs when a page is loaded in memory at the time the fault is generated, but is not marked in the MMU as being loaded in memory. The page fault handler in the OS merely needs to make the entry for that page in the memory management unit point to the page in memory and indicate that the page is loaded in memory; it does not need to read the page into memory. This could happen if the memory is shared by different programs and the page is already brought into memory for other programs.
3) A major (hard) page fault occurs when the page is not loaded in memory at the time of the fault. The page fault handler in the OS needs to find a free location: either a page in memory, or another non-free page in memory. This latter might be used by another process, in which case the OS needs to write out the data in that page (if it has not been written out since it was last modified) and mark that page as not being loaded in memory in its process page table. Once the space has been made available, the OS can read the data for

the new page into memory, add an entry to its location in the memory management unit (MMU), and indicate that the page is loaded. Thus major faults are more expensive than minor faults and add disk latency to the interrupted program's execution.

c) 1 kByte = 1000

$2^{10} = 1024$

10 bits for page offset

16 − 10 = 6 bits for pagenumber

∴ $2^6 = 64$ entries.

64 × 16 = 1024 bytes

∴ 1 kByte page table size

d i) page table:

modified bit, valid bit — 0 → no, 1 → yes

Index				
0	0010	1101	0000	0000
1	0000	0100	0000	0011
2	1101	1101	0000	0001
3	0000	0000	0000	0000
4	0111	1101	0000	0001

0x B85 → $\underbrace{0000\ 10}_{\text{index 2 into page table}}11\ 1000\ 0101$

index 2 is valid ∴ is translatable ∴ $\underbrace{1101\ 11}_{\text{from page table}}\underbrace{11\ 1000\ 0101}_{\text{offset is the same.}}$

ii) 0x1420 → $\underbrace{0001\ 01}_{\text{5 is not an index in the page table}}00\ 0010\ 0000$ → untranslatable − page fault

iii) 0x1000 → $\underbrace{0001\ 00}_{\text{index 4 into page table}}00\ 0000\ 0000$ → valid bit is set so can translate to

⇒ $\underbrace{0111\ 11}_{\text{from page table}}\underbrace{00\ 0000\ 0000}_{\text{offset the same}}$

iv) 0xC9A ⇒ $\underbrace{0000\ 11}_{\text{index 3}}00\ 1001$ 1010

page − fault since not valid

e) Evict page @ index 1 as both modified & valid bit is set. Some eviction policies flush modified pages regularly from page table