CS5250 Advanced Operating Systems

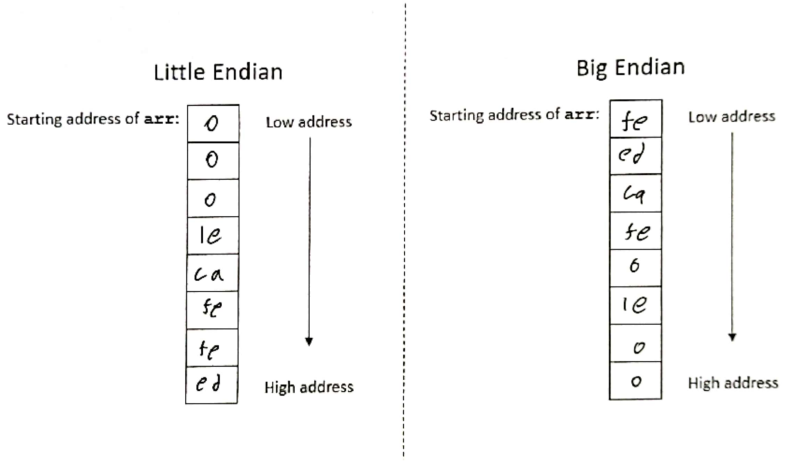
Pop Quiz 1

Name: Daniel Alfred

Student Number: A0184588J

1. Assuming integers are 4 bytes, show how memory is will be laid out for the following scenarios by filling in the content of memory (each box holds a byte):

```
int arr[2] = { 0x1e, 0xcafefeed};
```



2. Suppose the CS register now contains the value "0xabc", and the IP register contains the value "0x89". What is the value of the program counter, i.e., CS:IP, in the x86 real-address mode?

0x AC49

CS5250 Advanced Operating Systems

Pop Quiz 2

Name: Daniel Alfred

Student Number: A0184588J

Write the snippet of x86-64 assembly code that will compute the polynomial:

$$a = 3x^3 - 4x^2 + x - 7$$

where "a" is **%rax** and x is in the memory location pointed to by **%rbp**. Both a and x are 64-bit integers. You may assume that all registers other than **%rbp**, **%rsp** and **%rip** are available for your use. The final answer is to be left in **%rax**.

```
mov rbx, rbp
mov rax, -7
add rax, rbx
imul rbx, rbp
mov rcx, rbx
imul rcx, -4
add rax, rcx
imul rbx, rbp
imul rbx, 3
add rax, rbx
```

CS5250 Advanced Operating Systems

Pop Quiz 3

(Due: 1 Feb 2021, 11pm)

Name: Daniel Alfred_____

Student Number: A0184588J_____

1. Refer to slide 39 of Lecture slide deck 4 “Linking and Loading”.

Assuming that **maskwords** = 4, **shift2** = 7, and **C** = 64, compute **N**, and **BITMASK** (in hexadecimal) for the string “**printf**”. Use the hash function on the same slide. Write down any assumptions that you feel you needed to make.

h1 = 359345080

h2 = 2807383

n = 2

BITMASK = 1800000

2. (I have already mentioned this in the recording but I want to make sure it “sank in” for you.) For a general Bloom filter using a bit vector of m bits and k hashes, argue why:

a) If the Bloom filter returns “no, not in the set” for an element e, it must be that e is not in the set.
b) If the Bloom filter returns “yes, may be in the set” for an element e, e may (true positive) or may not (false positive) be in the set. In particular, what would be the worst case scenario for false positives?

a. The contraposition is if e is in the set, bloom filter will returns yes, it may be in the set. In this case it’s true since once e is in the set, it will change all the bit in the vector for every hashes to true. So once the bloom filter checks, it will return yes.
b. The worst case scenario is let say k=2 and we have 3 element {a, b, e}. Let h1(a) = x and h2(b) = y and h1(e) = y and h2(e) = x.
If we add a and b to the set and check for e, the bloom filter will return true even though it’s not there. Even worse, if the hash function is bad, let all element x will be hashed into 1 value, even with k=1, this will give false positive

CS5250 Advanced Operating Systems

Pop Quiz 4

(Due: 16 Feb 2021, 11pm)

Name: _____Daniel Alfred Widjaja_____

Student Number: __A0184588J_____

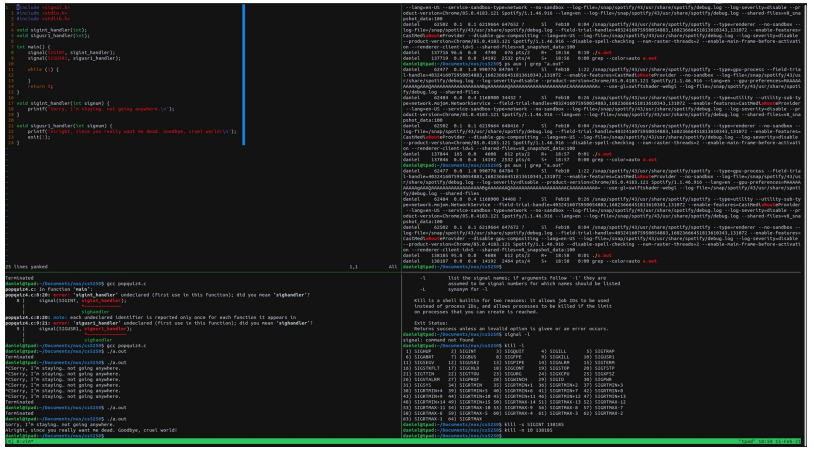
This is a small programming exercise on Linux signals.

Write a simple C program that satisfy the following specifications:

1. The main body performs an infinite loop.
2. Using the signal mechanism, when it receives a CTRL-C from the keyboard, the program will merely print out “Sorry, I’m staying… not going anywhere.” and continues with the infinite loop.
3. When it receives the signal 10, it will print out “Alright, since you really want me dead. Goodbye, cruel world!” and exits.

Show (by screen capture) how you it behaves. Hint: you may need two windows so that you can send the termination signal to the infinite loop program.

Submit your answer in a PDF into the corresponding Luminus submission folder.



```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void sigint_handler(int);
void sigusr1_handler(int);

int main() {
    signal(SIGINT, sigint_handler);
    signal(SIGUSR1, sigusr1_handler);

    while (1) {

    }
    return 0;
}

void sigint_handler(int signum) {
    printf("Sorry, I'm staying... not going anywhere.\n");
}

void sigusr1_handler(int signum) {
    printf("Alright, since you really want me dead. Goodbye, cruel world!\n");
    exit(1);
}
```

CS5250 Advanced Operating Systems

Pop Quiz 5

(Due: 3 Mar 2021, 11pm)

Name: _Daniel Alfred Widjaja_____

Student Number: A0184588J_____

Please do a code walkthrough of the Linux 5.10.6 kernel and explain how, starting with **`context_switch()`** in **`kernel/sched/core.c:3727`**, how context switching is achieved. In particular, staying on the 64 bit x86 architecture, trace the control flow and:

1. Identify the macros and procedures encountered, and try to explain what they do;
2. Identify the stacks involved and where the stack switches occur;
3. At key points where the control flow changes, where is on the top of the current stack? ("Key" is up to you to define but it should be used to clearly explain the flow that you have identified.)
4. Show how control will return back to the current task being switched out eventually.

Also, answer the following questions:

1. Why are RBX, RBP, R12-R15 pushed and then popped in **`__switch_to_asm`** (found in **`arch/x86/entry/entry_64.S`**)?
2. What is the effect of the do-while loop in the **`switch_to`** macro?

Submit your answer in a PDF into the corresponding Luminus submission folder.

1. The context is basically the task_struct, prev and next both contains the context and we want to switch the prev with the next. Rq is the request that

2. Stack swtiches occurs in line 3779.

3.In line 3731, the OS saving the states of prev, and preparing to switch.

In line 3740, itś basically decide how the switch is going to happened, depends on the type of protection (user mode or kernel mode).

In line 3776, preparing to switch in line 3779.

The finish_task_switch will do the cleanup

4. I think it could simply switch the prev and next.

1. I think these registers are used to support the context switching. And they may change while doing the switch. So restoring the value will ensure that they have a consistent value.

2. the do-while is used to wrap the statement. So the whole thing will work as one. So it couldn't be misread or misused by the statement before it.

CS5250 Advanced Operating Systems

Pop Quiz 6

Name: Daniel Alfred Widjaja

Student Number: A0184588J

Suppose there are three tasks, A, B, and C, of niceness levels -2, 0, 2, respectively. Assume that they arrived at the same time. Show the first 10 context switches that will happen under CFS and BFS. Please show your working, i.e., the intermediate state of the corresponding data structures. State any assumption clearly.

1. CFS:
    1. Starting vruntime = 0 for all.
    2. Load Weight:
        1. -2 = 1586
        2. 0 = 1024
        3. 2 = 655
        4. total = 3265
    3. Time share:
        1. -2 = 2914.54
        2. 0 = 1881.77
        3. 2 = 1203.67
    4. Sequence:
        1. -2, update vruntime += 1881.77
        2. 0, update vruntime += 1881.77
        3. 2, update vruntime += 1881.77
        4. -2
        5. 0
        6. 2
        7. -2
        8. 0
        9. 2
        10. -2
    5. At 4th, 7th, 10th step, the vruntime of all tasks will have the same value and will repeat the sequence.
2. BFS:
    1. Prioratio:
        1. -2 = 715
        2. 0 = 866
        3. 2 = 1048
    2. Scaling factor = s = 8192
    3. Niffies = t0, now for all

4. rr_interval = r = 6
5. Virtual Deadline
    1. -2 = t0 + 715 * s * r = 0 + 715 * 8192 * 6 = 35143680
    2. 0 = t0 + 866 * s * r = 0 + 866 * 8192 * 6 = 42565632
    3. 2 = t0 + 1048 * s * r = 0 + 1048 * 8192 * 6 = 51511296
6. Sequence:
    1. -2, update virtual deadline = 6000000 + 715 * s * r = 41143680
    2. -2, update virtual deadline = 12000000 + 35143680 = 47143680
    3. 0, vd = 18000000 + 42565632 = 60565632
    4. -2, vd = 24000000 + 35143680 = 59143680
    5. 2, vd = 30000000 + 51511296 = 81511296
    6. -2, vd = 36000000 + 35143680 = 71143680
    7. 0, vd = 42000000 + 42565632 = 84565632
    8. -2, vd = 48000000 + 35143680 = 83143680
    9. 2, vd = 54000000 + 51511296 = 105511296
    10. -2, vd = 60000000 + 35143680 = 95143680

CS5250 Advanced Operating Systems

Pop Quiz 7

Name: Daniel Alfred Widjaja_____

Student Number: A0185688J_____

First time trying this… hope I won't fall flat on my face.

This is an exercise to test if you understand virtual address translation.

In the same Luminous folder is a ZIP file. It contains a single Intel x86-64 Linux static executable called "pop7-linux-executable". I created it on a Centos machine and tested it on my Windows bash environment. It should work out of the box in a 64 bit Linux environment. It is a very simple executable that "pretends" to be a set of page tables. This assignment is individualized. So to start, input your student number. It will generate a unique virtual address for you. Then you are to use the page table simulator and input physical addresses to walk the page table and arrive at the final physical address for the virtual address you are given. Please provide the details of the steps involved in getting to the final result.

Caveat: the app is quite simple and dumb – input a page table address and it will output a fake physical address for the next level.

Virtual Address = 0x0e43ef48615c
binary :
PML4 = 000011100
Dir ptr = 100001111
Dir = 101111010
Table = 010000110
Offset = 000101011100

PML4E = table(CR3 + PML4 * 16 bits) = table(0x104d210f0) = 0x2cb03ec000
PDPTE = table(PML4E + DirPtr * 16) = table(0x2cb03ed0f0) = 0x6c02123000
PDE = table(PDPTE + Dir * 16) = table(0x6c021247a0) = 0x4a0db4e000
PTE = table(PDE + table * 16) = table(0x4a0db4e860) = 0x5cfb8dc000
Physical Address = 0x5cfb8dc000 + 0b000101011100 * 16 = 0x5cfb8dd5c0

CS5250 Advanced Operating Systems

Pop Quiz 8

Name: _Daniel Alfred Widjaja_____

Student Number: A0184588J_

Consider the following piece of pseudo code:

```
…
R1 ← <var X in memory>
R1 ← R1 + 1
<var X in memory> ← R1
…
```

Suppose there are two threads executing this code in parallel. Show what can go wrong. Then suppose there is a lock that is used locked before these three lines and unlocked afterwards. Show how the problem is fixed.

Thread 1	Thread 2
R1 ← X	
R1 ← R1 + 1	
	R1 ← X
	R1 ← R1 + 1
X ← R1	
	X ← R1

in the case above, even though we add R1 2 times, we only add the value by one in the end. This happens because thread 1 write is overwrite by thread 2.

If we have a lock, thread 1 write will happen before thread 2 read, or the other way around. That way, whichever reading later will read the value which already been incremented

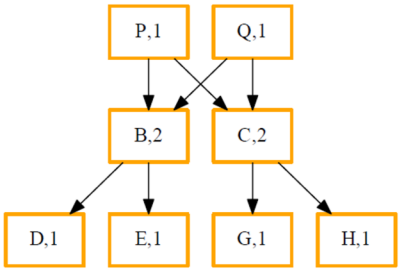
CS5250 Advanced Operating Systems

Pop Quiz 9

Name: Daniel Alfred Widjaja_____

Student Number: A0184588J_____

Assuming this initial btrfs tree:



Show the steps involved in the insertion of a new leaf node X that lies between E and G, along with the evolution of the reference counts, using the notations in the lecture slides. Clearly state any assumptions made.

- Inserting X, from the top, should've compared Q with X. if X < Q, we push X to B, otherwise we push X to C. Regardless, it will create (Q', 1) where it replaces (Q, 1) which becomes (Q, 0) and get deleted.
- Assuming pushed to B, (B, 2) will become (B, 1) because (Q, 1) is deleted, and we create (B', 1) where (Q', 1) is pointing there.
- Then because (B', 1) is copying B, it will point to (D, 1) and (E, 1). And because we have X > E, we create a new node on the right part of E, and name it (X, 1). because only (B', 1) is pointing there.

CS5250 - Assignment 1

Daniel Alfred Widjaja - A0184588J

Running virtual box

- Followed the instruction and the first problem I found is
- Before doing anything, this is the kernel version and MAC address.

```
daniel-alfred@monmouth:~$ uname -a
Linux monmouth 5.8.0-41-generic #46-Ubuntu SMP Mon Jan 18 16:48:44 UTC 2021 x86
_64 x86_64 x86_64 GNU/Linux
```

```
daniel@monmouth:~/work/linux-5.10.6$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 10
00
    link/ether 08:00:27:95:55:39 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 84828sec preferred_lft 84828sec
    inet6 fe80::1563:c176:fcf3:2b6e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
```

- **Task 2**: When running ‘make menuconfig’ there’s 3 options which are built-in, exclude, or module.
  - built-in means installed directly to the kernel
  - module means it’s installed as a module which can be removed if you wish
  - while exclude means it’s not installed at all
- The one that appears in the kernel image is built-in only.
- I did not change anything in /boot/grub/grub.cfg since it’s a generated file

```
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by grub-mkconfig using templates
# from /etc/grub.d and settings from /etc/default/grub
#
```

- I reboot and the kernel is updated

```
daniel@monmouth:~$ uname -a
Linux monmouth 5.10.6 #1 SMP Mon Feb 8 08:10:26 UTC 2021 x86_64 x86_64 x86_64 G
NU/Linux
```

- This happens because ‘make install’ already changes the grub.cfg for us as we can see in the image below.

```
daniel@monmouth:~/work/linux-5.10.6$ make install
sh ./arch/x86/boot/install.sh 5.10.6 arch/x86/boot/bzImage \
        System.map "/boot"
mv: cannot move '/boot/vmlinuz-5.10.6' to '/boot/vmlinuz-5.10.6.old': Permission denied
make[1]: *** [arch/x86/boot/Makefile:160: install] Error 1
make: *** [arch/x86/Makefile:275: install] Error 2
daniel@monmouth:~/work/linux-5.10.6$ sudo !!
sudo make install
[sudo] password for daniel:
sh ./arch/x86/boot/install.sh 5.10.6 arch/x86/boot/bzImage \
        System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 5.10.6 /boot/vmlinuz-5.10.6
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 5.10.6 /boot/vmlinuz-5.10.6
update-initramfs: Generating /boot/initrd.img-5.10.6
find: ‘/var/tmp/mkinitramfs_KUeoqP/lib/modules/5.10.6/kernel’: No such file or directory
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 5.10.6 /boot/vmlinuz-5.10.6
run-parts: executing /etc/kernel/postinst.d/update-notifier 5.10.6 /boot/vmlinuz-5.10.6
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 5.10.6 /boot/vmlinuz-5.10.6
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/init-select.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-5.10.6
Found initrd image: /boot/initrd.img-5.10.6
Found linux image: /boot/vmlinuz-5.10.6.old
Found initrd image: /boot/initrd.img-5.10.6
Found linux image: /boot/vmlinuz-5.8.0-41-generic
Found initrd image: /boot/initrd.img-5.8.0-41-generic
Found linux image: /boot/vmlinuz-5.8.0-25-generic
Found initrd image: /boot/initrd.img-5.8.0-25-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
```

Building smaller kernel

- By seeing ‘make help’ we can see there is ‘make tinyconfig’ which allows us to create the tiniest possible kernel.

```
Configuration targets:
  config          - Update current config utilising a line-oriented program
  nconfig         - Update current config utilising a ncurses menu based program
  menuconfig      - Update current config utilising a menu based program
  xconfig         - Update current config utilising a Qt based front-end
  gconfig         - Update current config utilising a GTK+ based front-end
  oldconfig       - Update current config utilising a provided .config as base
  localmodconfig  - Update current config disabling modules not loaded
                    except those preserved by LMC_KEEP environment variable
  localyesconfig  - Update current config converting local mods to core
                    except those preserved by LMC_KEEP environment variable
  defconfig       - New config with default from ARCH supplied defconfig
  savedefconfig   - Save current config as ./defconfig (minimal config)
  allnoconfig     - New config where all options are answered with no
  allyesconfig    - New config where all options are accepted with yes
  allmodconfig    - New config selecting modules when possible
  alldefconfig    - New config with all symbols set to default
  randconfig      - New config with random answer to all options
  yes2modconfig   - Change answers from yes to mod if possible
  mod2yesconfig   - Change answers from mod to yes if possible
  listnewconfig   - List new options
  helpnewconfig   - List new options and help text
  olddefconfig    - Same as oldconfig but sets new symbols to their
                    default value without prompting
  tinyconfig      - Configure the tiniest possible kernel
  testconfig      - Run Kconfig unit tests (requires python3 and pytest)
```

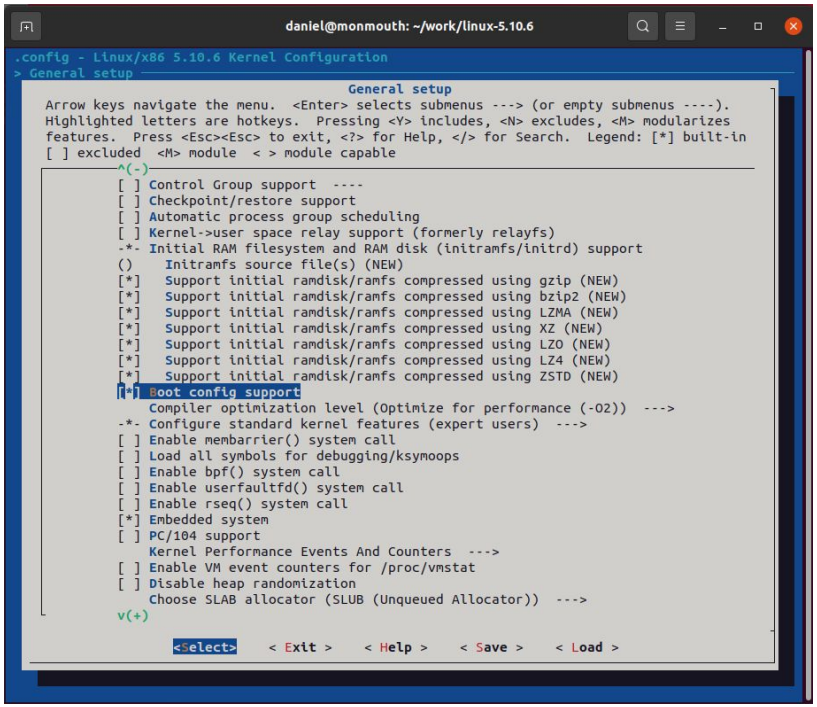
- Interesting thing happens when ‘make modules_install’ and it doesn’t install any modules.

```
daniel@monmouth:~/work/linux-5.10.6$ sudo make modules_install
[sudo] password for daniel:

The present kernel configuration has modules disabled.
Type 'make config' and enable loadable module support.
Then build a kernel with module support enabled.

make: *** [Makefile:1458: modules_install] Error 1
```

- However, after I reboot, it does not load.
- So I restart everything (because I forgot to take a snapshot)



- I started with tinyconfig again and include boot config support. It runs into grub but still can’t log in.

make localmodconfig

- So instead of making from tinyconfig, I start with the default config and disable the not needed modules. make localmodconfig allows us to disable modules not loaded.
  - The disabling module part is a little bit trial and error because I tried to disable more and more modules and if it doesn’t boot, I will revert to the previous version.
- Additionally we can disable certain stuff like:
  - several networking support
  - security option
  - use XZ compression

- etc.
- The excluded modules above can make the kernel image size smaller as can be seen below .

```
daniel@monmouth:~/work/linux-5.10.6$ du -h arch/x86/boot/bzImage
3.3M    arch/x86/boot/bzImage
```

- The kernel works and only cost 3.3MB.

```
daniel@monmouth:~$ uname -a
Linux monmouth 5.10.6 #1 Tue Feb 9 12:39:04 +08 2021 x86_64 x86_64 x86_64 GNU/L
inux
```

Part B

1. 4d 29 44 7a a7
    a. 4d is a prefix (0100 1101), W = 1, R = 1, X = 0, B = 1
    b. 29 is the opcode that means subtract
    c. 44 is the ModRM byte which is 0100 0100
        i. mod = 01 (with register, [r/m + disp8] is used
        ii. R + reg/opcode = 1000 (r8)
        iii. B + R/M = 1100
    d. 7a is the SIB byte which is 0111 1010
        i. scale = 01 = 2
        ii. X + Index = 0111 (rdi)
        iii. B + base = 1010 (r10), based on the table this makes [base + index * s]
        iv. so [r10 + rdi * 2]
    e. a7 is the displacement which is 1010 0111
        i. That is the binary of -0x59
        ii. 1010 0111 --inverse--> 0101 1000 --(+1)--> 0101 1001 = 0x59
    f. `sub    %r8, -0x59(%r10, %rdi, 2)`
2. `addl 8(%esp), %ecx`
    a. `addl = 03 opcode because we want to add 32 bits to 32 bits`
    b. `4c (0100 1100) is the ModRM, this is for %ecx`
        i. `mod = 01 (1 byte displacement)`
        ii. `reg ecx = 001`
        iii. `RM = 100 by default`
    c. `24 (0010 0100) is the SIB bytes, this is for %esp`
        i. `mod = 00 (no multiplier)`
        ii. `index illegal = 100`
        iii. `base esp = 100`
    d. `08 is the displacement`
        i. `+8 = 0x08`
    e. `03 4c 24 08`

```
3. void unknown_func(char *c) {
        int arr[10];
        arr[7] = 0;
        while (*c != 0) {
             if (*c == 101) break;
             arr[7]++;
             *c++;
        }
        arr[0] = arr[7];
        printf("%d\n", arr[7]);
   }
```

CS5250 - Assignment 2

Daniel Alfred - A0184588J

Part A

We could mount it to /sys/kernel/tracing, by using `mount -t tracefs nodev /sys/kernel/tracing`

There are a couple of parameters in the form of files that we could change in the folder.

```
daniel@tpad:~$ sudo !!
sudo ls /sys/kernel/tracing
available_events          enabled_functions         max_graph_depth      set_event_pid       stack_trace_filter  trace_stat
available_filter_functions  error_log               options              set_ftrace_filter   synthetic_events    tracing_cpumask
available_tracers         events                    per_cpu              set_ftrace_notrace  timestamp_mode      tracing_max_latency
buffer_percent            free_buffer               printk_formats       set_ftrace_pid      trace               tracing_on
buffer_size_kb            function_profile_enabled  README               set_graph_function  trace_clock         tracing_thresh
buffer_total_size_kb      hwlat_detector            saved_cmdlines       set_graph_notrace   trace_marker        uprobe_events
current_tracer            instances                 saved_cmdlines_size  snapshot            trace_marker_raw    uprobe_profile
dynamic_events            kprobe_events             saved_tgids          stack_max_size      trace_options
dyn_ftrace_total_info     kprobe_profile            set_event            stack_trace         trace_pipe
```

We could set the *current_tracer* to check what we are currently tracing. By default, it should have “nop” inside where it means there is nothing to trace. To check what we can trace, we could see the contents of *available_tracers*. Then we can echo what we want to trace to the *current_tracer* file.

There is a *tracing_on* file. This basically the file that controls whether we start or stop the tracing. We set 1 on this file if we want to start tracing and set it to 0 if we want to stop. There are also commands *traceon* and *traceoff* which do literally what it says, turning the trace on or off. The command could be set in the *set_ftrace_filter* file.

The trace is stored in the *trace* file. As we can see to the screenshot below, this is what we get if we open the file.

echo function_graph > current_tracer
echo 1 > tracing_on
head trace

```
root@tpad:/sys/kernel/tracing# head trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 0)   0.201 us    |                    rcu_all_qs();
 0)   0.573 us    |                  } /* _cond_resched */
 0)   0.985 us    |                } /* mutex_lock */
 0)               |                tty_write_room() {
 0)               |                  pty_write_room() {
 0)   0.205 us    |                    tty_buffer_space_avail();
```

You can set *set_graph_function* to filter which function to trace. For instance, if you only wants to filter *__do_fault*, we can

echo __do_fault > set_graph_function

We can limit the depth of the trace, by using

echo 2 > max_graph_depth

Part B

I realize that printk is printing to the kernel while printf is printing to file descriptor. So we will use printk because it’s available. printf can specify a loglevel, however the kernel uses loglevel to decide whether to print the message to the console.

I add printmsg.c file which contains the code in the assignment pdf. Then I add to the table in arch/x86/entry/syscalls/syscall_64.tbl

```
600     common  printmsg                  sys_printmsg
```

I add this line which allows us to call printmsg in both x86 and x32 with the number code of 600. Number 548+ are free to use.

Then after reinstalling the kernel, I reboot.

Task 4

1. No. 1
   a. clear the ‘trace’ file
      i. echo > trace
   b. *buffer_size_kb* will modify the number of entries that can be recorded.
      i. echo 10 > *buffer_size_kb*
      ii. This command will limit 10kilobytes for each CPU
   c.
```c
#include <linux/ftrace.h>

int main() {
    // trace printk will create a comment of
    // Hello World in the main() function.
    trace_printk("Hello World!");
}

```
      i. We can use dynamic trace to trace this specific function.
      ii. To run the shell, we need to do
         1. sudo su
         2. cd /sys/kernel/tracing
         3.
```
echo mmiotrace_printk > set_ftrace_filter
echo function > current_tracer
echo 1 > tracing_on
usleep 1
echo 0 > tracing_on
```
         4. run the shell script.

d.

```
root@tpad:/sys/kernel/tracing# head -n 20 trace
# tracer: function
#
# entries-in-buffer/entries-written: 204854/35068654   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID     CPU#  ||||   TIMESTAMP  FUNCTION
#              | |         |   ||||      |         |
          <...>-38132   [003] .... 37967.617275: __fdget <-ksys_ioctl
          <...>-38132   [003] .... 37967.617275: __fget_light <-__fdget
          <...>-38132   [003] .... 37967.617275: __fget <-__fget_light
          <...>-38132   [003] .... 37967.617275: security_file_ioctl <-ksys_ioctl
          <...>-38132   [003] .... 37967.617275: do_vfs_ioctl <-ksys_ioctl
          <...>-38132   [003] .... 37967.617275: VBoxDrvLinuxIOCtl_6_1_16 <-do_vfs_ioctl
          <...>-38132   [003] .... 37967.617275: supdrvIOCtlFast <-VBoxDrvLinuxIOCtl_6_1_16
          <...>-38132   [003] .... 37967.617275: VBoxHost_RTThreadNativeSelf <-0xffffa8c40567dc0b
          <...>-38132   [003] .... 37967.617275: VBoxHost_RTThreadPreemptDisable <-0xffffa8c40567dcc2
```

There are couple of things that we can see

i. There are 204854 entries in buffer and 35068654 events in total. So some of them is not in the trace file because it's gonna be too large.
ii. Task PID - The process ID that called the function
iii. The CPU ID that call the function
iv. Timestamp in seconds when the function was entered.
v. function name. There are 2 names, the first name is the name of the function and the second one is the parent function who called it. In the first example
    1. The function is __fdget,
    2. It's called from ksys_ioctl

2.

```
root@tpad:/sys/kernel/tracing# echo vfs_open > set_graph_function
root@tpad:/sys/kernel/tracing# echo vfs_read >> set_graph_function
root@tpad:/sys/kernel/tracing# echo vfs_write >> set_graph_function
root@tpad:/sys/kernel/tracing# cat set_graph_function
vfs_read
vfs_open
vfs_write
root@tpad:/sys/kernel/tracing# echo > trace
root@tpad:/sys/kernel/tracing# cat max_graph_depth
2
root@tpad:/sys/kernel/tracing# echo 10 > max_graph_depth
root@tpad:/sys/kernel/tracing# echo 1 > tracing_on
root@tpad:/sys/kernel/tracing# wc -n trace
wc: invalid option -- 'n'
Try 'wc --help' for more information.
root@tpad:/sys/kernel/tracing# wc -l trace
780 trace
root@tpad:/sys/kernel/tracing# wc -l trace
828 trace
root@tpad:/sys/kernel/tracing# wc -l trace
903 trace
root@tpad:/sys/kernel/tracing# wc -l trace
847 trace
root@tpad:/sys/kernel/tracing# echo 0 > tracing_on
root@tpad:/sys/kernel/tracing# head -20 trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 1)   2.265 us    |        fsnotify();
 1)   9.876 us    |      } /* security_file_permission */
 1) + 11.248 us   |    } /* rw_verify_area */
 1)               |    __vfs_read() {
 1)               |      eventfd_read() {
 1)   0.612 us    |        _raw_spin_lock_irq();
 1)   1.986 us    |      }
 1)   3.330 us    |    }
 1)   0.602 us    |    __fsnotify_parent();
 1)   0.600 us    |    fsnotify();
 1) + 20.343 us   |  } /* vfs_read */
 ------------------------------------------
 1)   gdbus-2050   =>   gmain-1015
 ------------------------------------------

 1)               |  vfs_write() {
root@tpad:/sys/kernel/tracing#
```

We can echo those 3 functions to track the tracing. Then set the max_graph_depth to 10 so that it can trace deeper.

```
3)               |  vfs_write() {
3)               |    rw_verify_area() {
3)               |      security_file_permission() {
3)               |        apparmor_file_permission() {
3)               |          common_file_perm() {
3)   0.705 us    |            aa_file_perm();
3)   1.867 us    |          }
3)   2.994 us    |        }
3)   4.279 us    |      }
3)   5.367 us    |    }
3)               |    __vfs_write() {
3)               |      eventfd_write() {
3)   0.646 us    |        _raw_spin_lock_irq();
3)               |        __wake_up_locked_key() {
3)               |          __wake_up_common() {
3)               |            pollwake() {
3)               |              default_wake_function() {
3)               |                try_to_wake_up() {
3)   0.632 us    |                  _raw_spin_lock_irqsave();
3)               |                  select_task_rq_fair() {
3)   1.012 us    |                    select_idle_sibling();
3)   2.216 us    |                  }
3)   0.611 us    |                  _raw_spin_lock();
3)   0.645 us    |                  update_rq_clock();
3)               |                  ttwu_do_activate() {
3)   9.391 us    |                    activate_task();
3)   1.760 us    |                    ttwu_do_wakeup();
3) + 12.828 us   |                  }
3)   0.581 us    |                  _raw_spin_unlock_irqrestore();
3) + 21.497 us   |                }
3) + 22.589 us   |              }
3) + 23.673 us   |            }
3) + 24.932 us   |          }
3) + 26.050 us   |        }
3) + 28.454 us   |      }
3) + 29.622 us   |    }
3)   0.604 us    |    __fsnotify_parent();
3)   0.590 us    |    fsnotify();
3) + 39.137 us   |  }
2)   0.570 us    |          } /* tty_hung_up_p */
2)               |          mutex_lock() {
2)               |            _cond_resched() {
2)   0.573 us    |              rcu_all_qs();
2)   1.645 us    |            }
2)   2.787 us    |          }
2)               |          pty_write() {
2)   0.622 us    |            _raw_spin_lock_irqsave();
2)               |            tty_insert_flip_string_fixed_flag() {
2)   0.907 us    |              __tty_buffer_request_room();
2)   2.087 us    |            }
```

As we can see here, cpu 3 is doing vfs_write, then cpu 2 continue doing it's tracing as well. Here they're doing the tracing at the same time

3. No 3
   a. Code:
      i.
```
daniel@monmouth:~/work/linux-5.10.6/kernel$ cat printmsg.c
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE1(printmsg, int, i)
{
        printk(KERN_DEBUG "Hello! This is A0184588J from %d", i);
        return 1;
}
```
      ii.
```
daniel@monmouth:~$ cat test.c
#include <linux/unistd.h>
#include <stdlib.h>
#define __NR_printmsg 600

int printmsg(int i) {
        return syscall(__NR_printmsg, i);
}

int main(int argc, char** argv)
{
        printmsg(atoi(argv[1]));
        return 0;
}
```
   b.
```
daniel@monmouth:~/work/linux-5.10.6/arch/x86/entry/syscalls$ pwd
/home/daniel/work/linux-5.10.6/arch/x86/entry/syscalls
```
   c.
```
daniel@monmouth:~$ ./a.out 9090
daniel@monmouth:~$ dmesg | tail
[    1.243045] systemd[1]: Mounted Kernel Trace File System.
[    1.252046] systemd[1]: modprobe@drm.service: Succeeded.
[    1.252440] systemd[1]: Finished Load Kernel Module drm.
[    1.258216] systemd[1]: Finished Uncomplicated firewall.
[    1.265964] systemd[1]: Started Journal Service.
[    1.269758] EXT4-fs (sda3): re-mounted. Opts: errors=remount-ro
[    1.576219] random: crng init done
[    2.856792] e2scrub_all (290) used greatest stack depth: 13432 bytes left
[    9.532707] hrtimer: interrupt took 4731021 ns
[   96.831746] Hello! This is A0184588J from 9090
```
   d. I installed the tiny kernel which makes the installation faster, but I realized that the available_tracer are missing. The only available one are only blk and nop, where we need to set the current_tracer to function. I need more time to install the whole kernel.

Part C

1. Let there are 2 hash functions h1 and h2 for the bloom filter function. And there is 4 data a, b, c, d.

Variable	h1	h2
a	x	y
b	x	z

- If we insert a to bloom filter, the set (bit-vector) would be {x, y}.
- Then we insert b to bloom filter, the set would be {x, y, z}.
- If I remove b from the bloom filter, the set will be {y}.
- However, when I check a, x is no longer there, so it results a wrong value.

2. no 2
   a.
      i. 3: 8b 15 00 04 fa aa
   b. rodata stores constant data. One should expect string literals, and other constant values to reside there. It is marked as read-only (although usually resides in a read and executable segment). I think to move the pointer in rodata file.
   c. So instead of having relocation, we can have a GOT which stores all the offset and when queried, it will point to the correct location.
      While PLT will handle function calls. I read from this website and it's quite clear how they explain the usage of GOT and PLT.
      https://www.technovelty.org/linux/plt-and-got-the-key-to-code-sharing-and-dynamic-libraries.html

Assignment 3

Part A

1. No 1.
   a. question a
      i. module_init is called when the module is installed
      ii. module_exit is called when the module is removed
   b. For:
      i. building the module using makefile

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

      ii. Installing the module using insmod hello.ko
      iii. Removing module using rmmod hello.ko
   c.

```
snowy@woof:~/work/modules$ sudo dmesg | tail
[    6.353887] 06:47:59.247842 main     6.1.16_Ubuntu r140961 started. Verbose l
evel = 0
[    6.369427] 06:47:59.263353 main     vbglR3GuestCtrlDetectPeekGetCancelSuppor
t: Supported (#1)
[    6.385009] 06:47:59.278828 automount vbsvcAutomounterMountIt: Successfully m
ounted 'cs5250' on '/home/snowy/cs5250'
[    6.895871] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
 RX
[    6.896251] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[   13.282839] rfkill: input handler disabled
[  582.264926] hello: loading out-of-tree module taints kernel.
[  582.264961] hello: module verification failed: signature and/or required key
missing - tainting kernel
[  582.265058] Hello, world
[  591.341818] Goodbye, cruel world
snowy@woof:~/work/modules$ F
```

d.

```
snowy@woof: ~/work/modules
[  582.264961] hello: module verification failed: signature and/or required key
missing - tainting kernel
[  582.265058] Hello, world
[  591.341818] Goodbye, cruel world
[ 1422.147450] Hello,
[ 1440.420814] Goodbye,
snowy@woof:~/work/modules$ insmod hello.ko who=A0184588J
insmod: ERROR: could not insert module hello.ko: Operation not permitted
snowy@woof:~/work/modules$ sudo !!
sudo insmod hello.ko who=A0184588J
snowy@woof:~/work/modules$ sudo rmmod hello.ko
snowy@woof:~/work/modules$ sudo dmesg | tail
[    6.896251] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[   13.282839] rfkill: input handler disabled
[  582.264926] hello: loading out-of-tree module taints kernel.
[  582.264961] hello: module verification failed: signature and/or required key
missing - tainting kernel
[  582.265058] Hello, world
[  591.341818] Goodbye, cruel world
[ 1422.147450] Hello,
[ 1440.420814] Goodbye,
[ 1473.708971] Hello, A0184588J
[ 1480.361443] Goodbye, A0184588J
snowy@woof:~/work/modules$
```

```
snowy@woof: ~/work/modules
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("GPL");

static char *who = "";
module_param(who, charp, 0660);

static int hello_init(void)
{
        printk(KERN_ALERT "Hello, %s\n", who);
        return 0;
}
static void hello_exit(void)
{
        printk(KERN_ALERT "Goodbye, %s\n", who);
}
module_init(hello_init);
module_exit(hello_exit);
```

2.

a. Since my onebyte is in 61, my mknod command is
mknod /dev/onebyte c 61 0

b.

```
snowy@woof:~$ ls -l /dev | grep onebyte
crw-r--r--  1 root  root     61,   0 Mar 21 19:16 onebyte
```

c.

```
ssize_t onebyte_read(struct file *filep, char *buf, size_t
        count, loff_t *f_pos)
{
    if( *f_pos >= 1 )
        return 0;
    /* If a user tries to read more than we have, read only as many by
tes as we have */
    if( *f_pos + count > 1 )
        count = 1 - *f_pos;
    if( copy_to_user(buf, onebyte_data + *f_pos, count) != 0 )
        return -EFAULT;
    /* Move reading f_pos */
    *f_pos += count;
    return count;
    //It was originally published on https://www.apriorit.com/
}
ssize_t onebyte_write(struct file *filep, const char *buf,
        size_t count, loff_t *f_pos)
{
    if( *f_pos >= 1 )
        return 0;
    /* If a user tries to write more than we have, read only as many b
ytes as we have */
    if (count != 1) {
        return -ENOSPC;
    }

    *onebyte_data = *buf;
    return count;
    //It was originally published on https://www.apriorit.com/
}
```

```
root@woof:/home/snowy/work/modules# cd
root@woof:~# cat /dev/onebyte
Xroot@woof:~# printf a> /dev/onebyte
root@woof:~# cat /dev/onebyte
aroot@woof:~# printf b> /dev/onebyte
root@woof:~# cat /dev/onebyte
broot@woof:~#printf zxc> /dev/onebyte
bash: printf: write error: No space left on device
root@woof:~# cat /dev/onebyte
broot@woof:~#
```

Part B

1. Processes:
    a.
        i. P1 burst CPU time: 23, arrival time 0
        ii. P2 burst CPU time: 12, arrival time 5
        iii. P3 burst CPU time: 41, arrival time 10
        iv. P4 burst CPU time: 17, arrival time 15
        v. P5 burst CPU time: 29, arrival time 40

All the time here is before context switching

time	P1	P2	P3	P4	P5
0	23	-	-	-	-
5	18	12	-	-	-
10	18	8	41	-	-
15	18	3	41	17	-
18	18	0	41	17	-
36	18	0	41	0	-
40	15	0	41	0	29
55	0	0	41	0	29
85	0	0	41	0	0
127	0	0	0	0	0

P1 finished in 55 seconds
P2 finished in 18 seconds
P3 finished in 127 seconds
P4 finished in 36 seconds
P5 finished in 85 seconds

b.

time	P1	P2	P3	P4	P5	Queue
0	23	-	-	-	-	
5	18	12	-	-	-	1, 2
10	13	12	41	-	-	2, 3, 1
15	13	7	41	17	-	2, 3, 1, 4
20	13	2	41	17	-	3, 1, 4, 2
30	13	2	31	17	-	1, 4, 2, 3
40	3	2	31	17	29	4, 2, 3, 1, 5
50	3	2	31	7	29	2, 3, 1, 5, 4
52	3	0	31	7	29	3, 1, 5, 4
62	3	0	21	7	29	1, 5, 4, 3

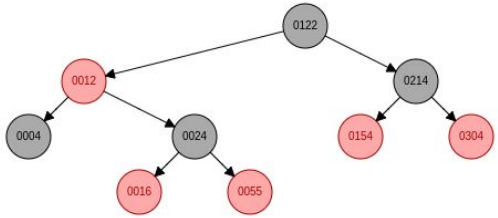
65	0	0	21	7	29	5, 4, 3
75	0	0	21	7	19	4, 3, 5
82	0	0	21	0	19	3, 5
92	0	0	11	0	19	5, 3
102	0	0	11	0	9	3, 5
112	0	0	1	0	9	5, 3
121	0	0	1	0	0	3
122	0	0	0	0	0	

P1 finished at 65 seconds.
P2 finished at 52 seconds.
P3 finished at 122 seconds.
P4 finished at 82 seconds.
P5 finished at 121 seconds.

2.

a. SJF will be the same as FCFS if all the jobs are the same.
So in a non-preemptive case and all the lengths are the same, as long as there is no job waiting, any scheduling will have the same minimizing the average response time.
No matter how you schedule the task, the number of tasks waiting for a particular time t[i] will always be the same in this case.
Since like the average response time is the sum of response time, moving around the order will only move the response time from process a to process b.
But the sum will remain the same.

b. Since the SRTF will run the shortest jobs and the lengths are the same, the shortest job will be the one who is already running. If there is no program running, then all the jobs will have the same length.



3. https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

CS5250 - Assignment 4

Daniel Alfred Widjaja - A0184588J

Section A

- I implemented the buddy allocator with 2-LRU list.
- There are 3 operation that could be implemented
    - Allocate
        - I put all the pages in the LRU inactive list
        - Remove all evicted pages from buddy system
        - Try to put everything continuously in the buddy system
        - If there's no space, I will allocate the page by page in the buddy system
    - Access
        - 3 possible cases:
            - The page is in inactive list
                - Remove from inactive list
                - Insert to active list
            - The page is in active list
                - Nothing to do
            - The page is not in both
                - Will insert to LRU and evict a page if needed.
                - If any page is evicted, then remove from buddy allocator
                - Insert to buddy system
    - Deallocate
        - If the page is in inactive list
            - Remove from inactive list
            - Remove from buddy allocator
        - If the page is in active list
            - Remove from active list
            - Remove from buddy allocator
        - If the page is not in both
            - Nothing to do
- The output format will consist of 3 parts,
    - Buddy allocator
        - The output will have
            - Page ID
                - The id of the occupied page
            - Group ID
                - Page with the same group ID are merged together
            - Length of group
                - The number of page with the same group ID
            - Seqno
                - num
    - Inactive LRU
        - The output will have
            - Index
                - Index of the element in the list
                - if the list is full, will evict the one with id 0
            - seqno
            - num
    - Active LRU
        - The output will have
            - Index
                - Index of the element in the list
                - if list is full, will move element with id 0 to inactive list.
            - seqno
            - num

Section B

1.
    a. Page0 = 0 | 0 | 4 = 4
    b. Page1 = 2 | 63 | 1 = 2 * 64 * 64 + 63 * 64 + 1 = 12225
    c. Page2 = 63 | 62 | 1 = 63 * 64 * 64 + 62 * 64 + 1 = 262017
2.
    a. init int *shared_ptr = 0;
    b. spin_lock() {
        i. while (!cas(shared_ptr, 0, 1)) {

            }
    }
    c. spin_unlock() {
        i. cas(ptr, 1, 0);
            }
    }

NATIONAL UNIVERSITY OF SINGAPORE

Semester 2 AY2017/18

CS5250 – ADVANCED OPERATING SYSTEMS

Time allowed: 2 hours

DRAFT ANSWERS

WRITE YOUR STUDENT NUMBER IN THESE BOXES

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **FIVE (5)** questions and comprises **FOURTEEN (14)** printed pages including this page.
2. This is an **OPEN BOOK** assessment.
3. Answer *all* questions. Note that the full mark for each question is different.
4. Write your answers on *this* **QUESTION AND ANSWER SCRIPT**. Answer only in the space (the box) given. Any writing outside this space will *not* be considered.
5. Fill in your Student Number with a pen, clearly on every page of this QUESTION AND ANSWER SCRIPT.
6. You may use pencil to write your answers.
7. At the end of the assessment, please check to ensure that your script has all the pages properly stapled together.
8. Note that when a number is written as "0xNNNN" it means that "NNNN" is in base 16.
9. Approved calculators are allowed for this assessment.

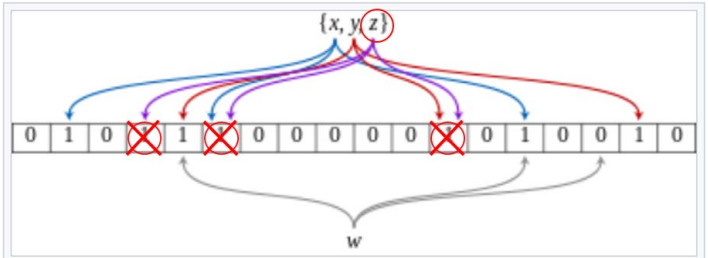
For Examiner's use only

Q1	
Q2	
Q3	
Q4	
Q5	
TOTAL	

QUESTION 1 (15 marks)

(1a) Using an example, show why deletion is not supported in the bit-vector form of the Bloom filter. (5 marks)

ANSWER:



An example of a Bloom filter, representing the set $\{x, y, z\}$. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set $\{x, y, z\}$, because it hashes to one bit-array position containing 0. For this figure, $m = 18$ and $k = 3$.

There are many possible answers but let's use this one. Deleting **z** will result in the three bit positions turned to zero. Since some of these positions are shared by **x** and **y**, any subsequent query of **x** and **y** will return "not found" – which is incorrect.

(1b) Suppose the single bit for each entry in a Bloom filter is replaced by an (unsigned) byte. Show what modifications to the algorithm you can make so as that deletion is now possible. (10 marks)

ANSWER:

For a counting Bloom filter: on insert, increment the hashed entries using saturating addition – if it hits the largest number – by one, it remains at that number. On deletion, for every hashed entry, decrement by 1 – again using saturating subtraction, i.e., if x = 0, then x – 1 = 0. An object is present if all the hashed locations are non-zero. In this way, deleting an object merely decrement the hash entries. If other objects are hashed to the same entries, then the count would be greater than 1 and hence avoid the situation in (1a).

QUESTION 2 (20 marks)

(2a) Let's assume that we have the following processes entering the system (in the given order):

- P1 burst CPU time: 23, arrival time 0
- P2 burst CPU time: 12, arrival time 5
- P3 burst CPU time: 41, arrival time 10
- P4 burst CPU time: 17, arrival time 15
- P5 burst CPU time: 29, arrival time 40

What is the execution schedule and the completion time for each if we schedule the processes using *pre-emptive shortest remaining time first* with a context switching overhead of 1 time unit? (10 marks)

ANSWER:

1. Time 1-5: P1 runs
2. P2 arrives at time 5, it has shorter remaining time so it gets to pre-empt P1. P1 has 18 units remaining.
3. Time 6: preemption
4. Time 7-18: P2 runs to completion.
5. In the meantime, at time 10, P3 arrives with 41 units and at time 15, P4 arrives at 17 units. However, both are longer than P2, so does not get to preempt P2.
6. Time 19: P4 is the shortest remaining time of 17 units and gets to run after P2 completes.
7. Time 20-36: P4 runs to completion.
8. Time 37: remaining tasks are P1 and P3. Since P1 has 18 units remaining. It gets to run.
9. Time 38-55: P1 runs to completion. In the meantime, P5 arrives but both P3 and P5 require longer time.
10. Time 56: P5 is the shorter of the two and gets to run.
11. Time 57-85: P5 runs to completion.
12. Time 86: remaining task P3 gets to run.
13. Time 87-127: P3 runs to completion.

P1: starts at time 0 and completes at time 55.
P2: starts at time 5 and completes at time 18.
P3: starts at time 10 and completes at time 127.
P4: starts at time 15 and completes at time 36.
P5: starts at time 57 and completes at time 85.

(2b) What is the completion time for each task if we schedule the processes using round robin with a quantum of 10 with no overhead in context switching?

(10 marks)

ANSWER:

1. Time 0-9: P1
2. Time 10-19: P2
3. Time 20-29: P3
4. Time 30-39: P4
5. Time 40-49: P5
6. Time 50-59: P1
7. Time 60-61: P2 - completes
8. Time 62-71: P3
9. Time 72-78: P4 - completes
10. Time 79-88: P5
11. Time 89-91: P1 - completes
12. Time 92-101: P3
13. Time 102-110: P5 - completes
12. Time 111-120: P3
13. Time 121: P3 - completes

P1: starts in time 0 and completes in time 91.
P2: starts in time 5 and completes in time 61.
P3: starts in time 10 and completes in time 121.
P4: starts in time 15 and completes in time 78.
P5: starts in time 40 and completes in time 110.

QUESTION 3 (30 marks)

(3a) Consider the following compare-and-exchange atomic instruction (abstracted as a function – in other words, assume that the following function is atomic):

```
int cas(int *ptr, int oldval, int newval);
```

If the integer pointed to by **`ptr`** is equal to **`oldval`**, then the content pointed to by **`ptr`** will be replaced with **`newval`**, and a '1' will be returned by **`cas`**. Otherwise, a '0' is returned. Using this function/instruction, show how a spinlock can be implemented by giving the C code for the **`spin_lock()`** and **`spin_unlock()`** functions. Write down any assumptions that you make. (15 marks)

ANSWER:

```
// assume this is the default at lock creation
#define UNLOCK 0

// Assume this is long enough for a process ID
typedef int spinlock_t;

int spin_lock(spinlock_t *lock)
{
    int pid = (int)getpid(); // assume PID cannot be 0

    while (!cas((int *)lock, UNLOCK, pid))

    return 1;
}

int spin_unlock(spinlock_t *lock)
{
    int curval = (int) *lock;
    int pid = (int)getpid(); // assume PID cannot be 0

    if (curval != pid) // Not current holder
       return -1;

    *lock = (spinlock_t) 0;
    return 1;
}
```

The lock is 0 if it is unlocked. If locked, it contains the process ID of the lock holder. This is for the unlocking procedure to check if the task attempting the unlock does indeed hold the lock.

(3b) Explain why it would not be straightforward to implement semaphores using basic `cas` instructions. What other facilities would you need? (5 marks)

ANSWER:

Semaphores involve blocking which transits a task from RUNNABLE to NON_RUNNABLE. This involves the scheduler API plus recording who is waiting against the semaphore so that a task can be waken up when the semaphore is "up". Hence, it is not merely the use of `cas`.

(3c) Suppose we implement the "big reader locks" using the idea outlined in class of per-CPU locks where each reader only needs to get one lock but writers need to acquire all locks. Suppose we have 4 CPUs, i.e., 4 locks. Show a scenario where we have two writers attempting to acquire all locks but resulting in a deadlock. (5 marks)

ANSWER:

Let the two writers be W1 and W2, and the 4 locks be L1, L2, L3, L4. What can happen is that W1 acquires L1, L2, L3, L4 in that order while W2 acquires L4, L3, L2, L1 in that order. Suppose W1 succeeds in acquiring L1 and L2 while W2 succeeds in acquiring L4 and L3. But now, W1 will wait for L3 – which holder, i.e., W2, will wait for L2. This results in a cycle of waits – and a deadlock.

(3d) What is the remedy to fix the problem you identified in **(3c)**? (5 marks)

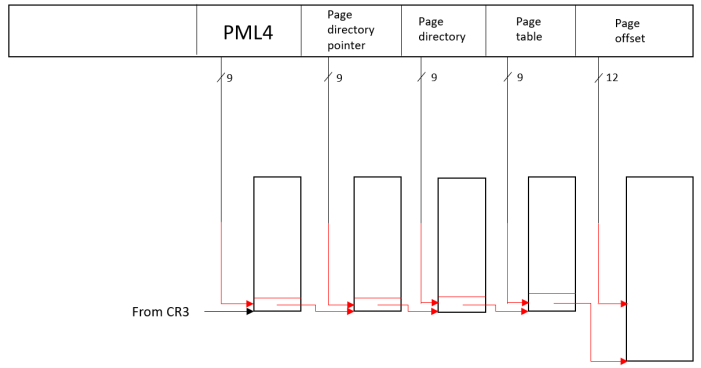
ANSWER:

This is a well known problem. Both W1 and W2 must acquire the locks in the SAME order. If so, only one will succeed in acquiring all locks while the other won't. The order of release (unlock) does not matter.

QUESTION 4 (20 marks)

(4a) Intel's 64 bit architecture uses 9 bits for PML4, page directory pointer, page directory and page table, and 12 bits for page offsets. Suppose a process has only 1 page from address 0 to 4095. Complete the following diagram by completing where the arrows will point to for a virtual address of 0. (5 marks)

ANSWER:



Since the address is all in the page offset bits, all the other bits are zero. So it will be the zero'th entry and the zero'th table every where else.

(4b) How much total memory would be needed for (4a)? Include the last one page that contains data. (5 marks)

ANSWER:

Simply, $(2^9 \times 4 \times 8) + 2^{12} = 20{,}480$ bytes.

(4c) Recalculate the memory required if all the 9-bit quantities in (4a) is now 10 bits. The process still only has a single page of data. (5 marks)

ANSWER:

$(2^{10} \times 4 \times 8) + 2^{12} = 36{,}864$ bytes.

(4d) What would be the advantages and disadvantages of going from 9 bits to 10 bits as done in (4c)? (5 marks)

ANSWER:

Advantages:

1. It increased the total physical address space from 48 to 52, making the addressable physical memory limit to go up from 2^{48} = 256TB to 2^{52} = 1024TB.

2. The same four tables (whose size increased from 16K bytes to 32K bytes) can accommodate 4,096 pages instead of 2,048 pages.

Disadvantages:

1. For a process that uses only a very small number of pages, there is a significant increase in overhead.

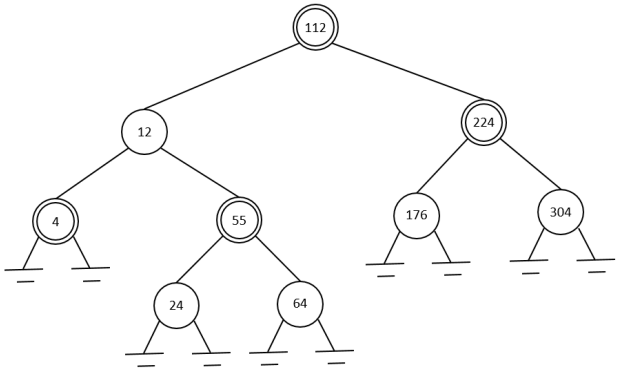
QUESTION 5 (15 marks)

(5a) Draw the final red-black tree after the following insertions are completed. Use a single circle to represent a "red" node and a double concentric circle to represent a "black" node. If you use other notations, make sure you give a legend.

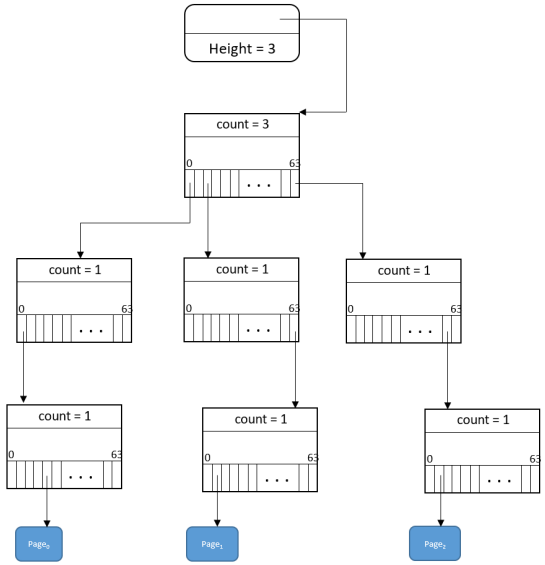
- Insert 112
- Insert 12
- Insert 304
- Insert 4
- Insert 55
- Insert 176
- Insert 64
- Insert 224
- Insert 24

(10 marks)

ANSWER:



(5b) Given the following Linux radix tree, what are the indexes of the three pages (in base 10)?



(5 marks)

Index of $Page_0$ in the diagram = 4_{10}

Index of $Page_1$ in the diagram = $12,225_{10}$

Index of $Page_2$ in the diagram = $262,017_{10}$

=== END OF PAPER ===

NATIONAL UNIVERSITY OF SINGAPORE

Semester 2 AY2018/19

CS5250 – ADVANCED OPERATING SYSTEMS

Time allowed: 2 hours

DRAFT OF ANSWERS

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **FIVE (5)** questions and comprises **SIXTEEN (16)** printed pages including this page.
2. This is an **OPEN BOOK** assessment.
3. Answer *all* questions. Note that the full mark for each question is different.
4. Write your answers on *this* **QUESTION AND ANSWER SCRIPT**. Answer only in the space (the box) given. Any writing outside this space will *not* be considered.
5. Fill in your Student Number with a pen, clearly on the front page and at least a few of the pages (at the top right and on all pages if possible) of this script.
6. You may use pencil to write your answers.
7. At the end of the assessment, please check to ensure that your script has all the pages properly stapled together.
8. Note that when a number is written as "0xNNNN" it means that "NNNN" is in base 16.
9. Approved calculators are allowed for this assessment.

For Examiner's use only

Q1	/15
Q2	/30
Q3	/20
Q4	/20
Q5	/15
TOTAL	/100

QUESTION 1 (15 marks)

(1a) The GNU Hash Bloom filter is uses the following code (as give in class):

- `H1 = dl_new_hash(symbol_name)`
- `H2 = H1 >> shift2`
- `N = ((H1 / C) % maskwords)`
- `BITMASK = (1 << (H1 % C)) | (1 << (H2 % C))`
- `bloom[N] |= BITMASK`
- `Test: (bloom[N] & BITMASK) == BITMASK`

Explain (i) why this is a k=2 Bloom filter, and (ii) how does it compare to the k=3 example given in class (slide 38 of Lecture Note set 4).

(10 marks)

ANSWER:

(i)

The bits at position H1 and H2 are the two hashed bits that are checked.

(ii)

The bit array (bloom[]) is accessed using a hash. So first hash is used to access the array, then the other 2 are used to check. So the hash is used 3 times, like in k=3 bloom filter using a single bit vector,

(1b) Explain the role of the global offset table (GOT) and procedure linkage table (PLT) in dynamic linking, especially at runtime. Why must some procedures be called via the PLT instead of directly in the code? (5 marks)

ANSWER:

The GOT is used for runtime resolution of external symbols, symbols whose final address in the virtual memory space is unknown at compile time. It is a table of addresses that will be fixed up by the runtime loader. The PLT contains code that will either call the dynamic loader (via the GOT) or the external routine directly to the routine (also via the GOT) once it has been resolved. Only local procedures whose addresses in the virtual memory space – either in absolute form or relative to some know location such as the current PC – can be called directly.

QUESTION 2 (30 marks)

(2a) Refer to the Linux load weight table given below:

```
static const int prio_to_weight[40] = {
    /* -20 */     88761,     71755,     56483,     46273,     36291,
    /* -15 */     29154,     23254,     18705,     14949,     11916,
    /* -10 */      9548,      7620,      6100,      4904,      3906,
    /*  -5 */      3121,      2501,      1991,      1586,      1277,
    /*   0 */      1024,       820,       655,       526,       423,
    /*   5 */       335,       272,       215,       172,       137,
    /*  10 */       110,        87,        70,        56,        45,
    /*  15 */        36,        29,        23,        18,        15,
};
```

Suppose there are 5, and only 5, tasks labelled A to E, at nice levels -2, -1, 0, 1, 2, respectively, in the system. Compute their respective share of CPU time. (10 marks)

ANSWER:

Total weight of system = 1586 + 1277 + 1024 + 820 + 655 = 5362.

Share of Task A: 1586 / 5362 = 30%

Share of Task B: 1277 / 5362 = 24.8%

Share of Task C: 1024 / 5362 = 19.1%

Share of Task D: 820 / 5362 = 15.3%

Share of Task E: 655 / 5362 = 12.2%

(2b) Compute the period for each task under Linux CFS under the same assumptions as in **Q2a**. (10 marks)

ANSWER:

Total weight of system = 1586 + 1277 + 1024 + 820 + 655 = 5362.

Assuming base if the default of 6ms, we have:

Period for Task A: 6 * 1586 / 5362 = 1.77ms

Period for Task B: 6 * 1277 / 5362 = 1.43ms

Period for Task C: 6 * 1024 / 5362 = 1.15ms

Period for Task D: 6 * 820 / 5362 = 0.92ms

Period for Task E: 6 * 655 / 5362 = 0.73ms

(2c) The actual **`vruntime`** update formula under Linux CFS is:

`vruntime +=` (physical time ran) * (weight of nice 0 task) / (weight of task)

Describe how the priority of a task influences the scheduling decision. (5 marks)

ANSWER:

This formula advances vruntime by a scaled value of the actual time the task ran. Because the weight of the higher priority tasks is larger, the actual increment in vruntime will be smaller for a higher priority task, giving it a higher chance of being to the left of the RB tree of tasks. Conversely, lower priority tasks will have their vruntime increment exaggerated in line with their weights, making them more likely to move to the right.

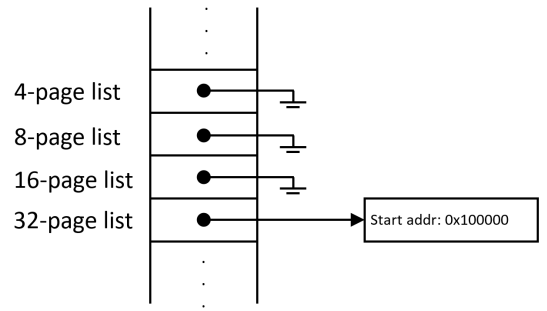
(2d) Suppose there are only two compute-only tasks in a single CPU system that runs indefinitely and never sleeps, one having the highest priority and the other having the lowest. How does CFS ensure that both tasks get a "fair" share of CPU time? In particular, will lower priority task ever to execute? (5 marks)

ANSWER:

The high priority task gets to run most of the time and for a long time. Each time it's current period expires, its vruntime gets updated. Meanwhile the vruntime of the low priority task remains the same. Eventually the vruntime of the high priority task will be greater than that of the low priority task – which gives the low priority task an opportunity to run, albeit for a short period. This way, even the lowest priority task will eventually get some (small) share of the CPU.

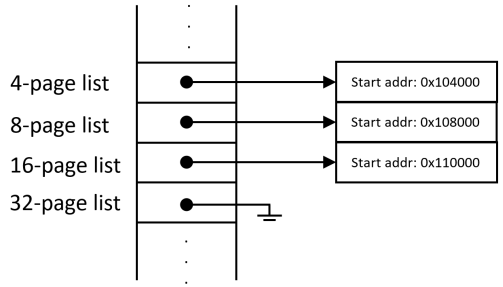
QUESTION 3 (20 marks)

(3a) Suppose the following is the current state of the buddy allocation list:



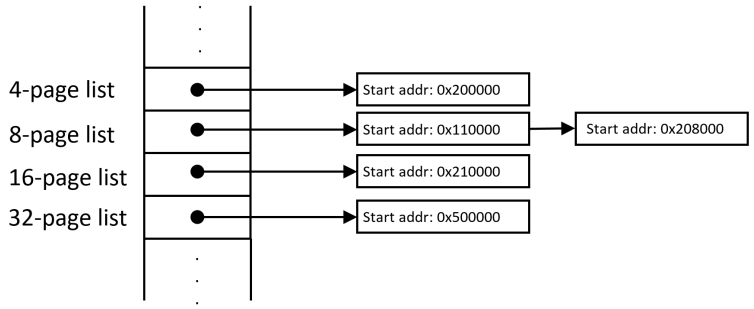
Draw the state of the lists after the allocation request for a block of 4 pages is satisfied. You are required to state the starting address of each block in the lists as well as give the starting address of the block returned. (5 marks)

ANSWER:



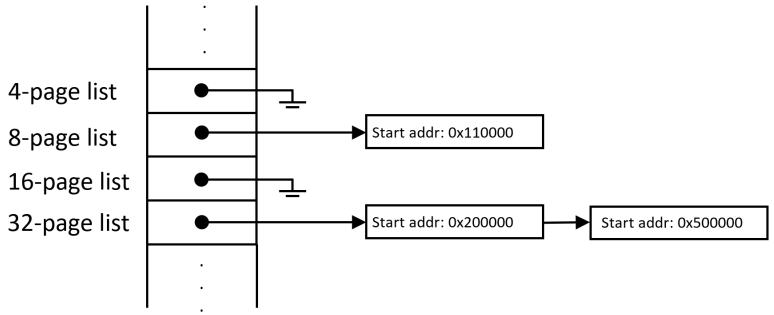
Starting address of returned block: 0x100000 (will also entertain 0x104000, 0x108000, 0x10c000 as correct answers since it is not specified which of the 4-page sub-blocks may be returned.

(3b) Suppose the following is the current state of the buddy allocation list:



Draw the state of the lists after a free request for a block of 4 pages with the starting address of **`0x204000`** is completed. You are required to state the starting address of each block in the lists. (5 marks)

ANSWER:



(3c) Under what circumstances is it (i) possible and (ii) impossible for a virtual address to miss in the TLB but at the same time hit in one of the data caches? (10 marks)

ANSWER:

(i)

If a data cache is large enough and the TLB is too small or busy, then an address could be cached into the data cache but its translation may not be cached in the TLB. What is important is that the translation via the page tables must be a valid one.

(ii)

What would be impossible would be a situation where the data is in the data cache, we have a TLB miss **AND** the translation via the page table is invalid. A page table translation is invalid means that the correspondence page(s) were never properly allocated in the process' virtual address space. If so, how can it be that valid data from a non-existent virtual page is cached?

QUESTION 4 (20 marks)

(4a) The size of largest file in ext2 format is actually limited by the 32-bit **`i_blocks`** field in the inode that represents the number 512-byte sector (confusingly also called "blocks") that a file can hold when in actual fact a data block in ext2 can be 1KB, 2KB, 4KB or 8KB. Now suppose we ignore the **`i_blocks`** field. What is the largest file that an ext2 file system using 4KB blocks can theoretically be, assuming pointers are 32-bits long? Show your working. (10 marks)

ANSWER:

The first 12 pointers point directly to 4KB blocks, i.e., the maximum would be 12 * 4KB = 48KB.

The 13th block points to a 4KB block of addresses. Since addresses are 32-bits or 4 bytes, there are 1024 pointers. Each of these points to a 4KB block. Hence we have 4096KB, or 4MB.

The 14th block has two levels of indirections. So there are 1024 * 1024 pointers to 4KB blocks, i.e., the maximum would be 4GB of storage.

The 15th block has three levels of indirections. So the maximum is $(1024)^3$ pointers to 4KB blocks, i.e., 4096GB of storage.

In total, we get (4096GB + 4GB + 4MB + 48KB) or well over 4TB for the maximum size of a file.

(4b) In a journaling file system, the journal is also written to the disk. So what happens if in the middle of writing the journal entry to disk, there is a failure? Explain how it is still possible for a journaling file system to recover itself. (5 marks)

ANSWER:

The key is that it must be possible to derive a consistent state of the filesystem during recovery. The operations to be executed on the file blocks are first recorded in the journal. Only after that is done successfully will the file blocks be changed. Suppose failure happens in the first step. Then the data in the file has not been changed and remains consistent. Suppose failure happens in the second step, then the journal is intact and the operations can be replayed to get the file blocks into a consistent state. If failure happens during the recovery process, the entire recovery process can simply be repeated.

(4c) Hard disks and SSDs guarantee that the write of a 512-byte sector is atomic. But a journal entry itself may be longer than 512-bytes. How can the atomic nature of the sector write be used to guarantee the integrity of journal entries? (5 marks)

ANSWER:

A journal entry will likely exceed 512-bytes. In order to guarantee the integrity of the entries, a special marker can be used to mark the end of an entry. If during the recovery process, an journal entry that has its end marker intact must have been completely written to the disk.

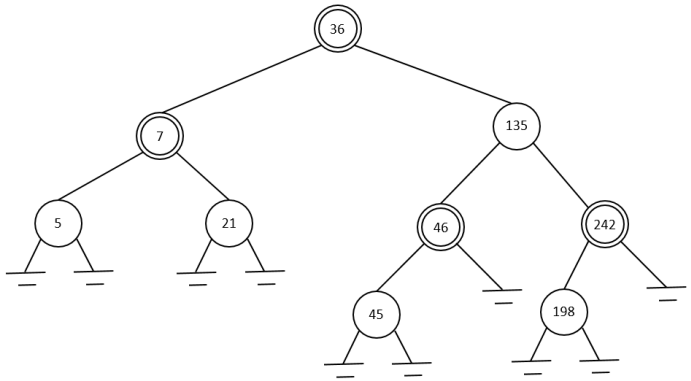
QUESTION 5 (15 marks)

(5a) Draw the final red-black tree after the following insertions are completed. Use a single circle to represent a "red" node and a double concentric circle to represent a "black" node. If you use other notations, make sure you give a legend.

- Insert 5
- Insert 135
- Insert 36
- Insert 46
- Insert 21
- Insert 242
- Insert 7
- Insert 198
- Insert 45

(10 marks)

ANSWER:



(5b) In Linux, exceptions are executed using the kernel stack in the user *process* context but interrupts are executed using the kernel's hard IRQ stack with a special context that is associated with a CPU rather than a process. What do you think are the design considerations for doing this?

(5 marks)

ANSWER:

Exceptions are errors generated usually from executing user code. As such, it makes sense to handle the exception quickly within the user context and send a signal to the same process.

=== END OF PAPER ===

Assignment 3	34/35	—	-1 - RB tree "nil" leaves not optional.
Pop Quiz 9	7/10	—	Need to see it diagrammatically.
Pop Quiz 8	9/10	—	This is called a "race condition". See: https://en.wikipedia.org/wiki/Race_condition
Pop Quiz 7	7/10	—	Why "* 16 bits"? Each entry is 8 bytes.
Pop Quiz 6	10/10	—	Answer correct except that the tasks are "A", "B" and "C".
Assignment 2	26/30	—	C.2 part a is incorrect. The rest not well answered.
Pop Quiz 5	8/10	—	Could have given more details.
Assignment 1	30/30	—	Perfect!
Pop Quiz 4	9/10	—	-1 for screen capture having too small fonts.
Pop Quiz 3	6/10	—	Size of one bitmap (C) is 64. Also, worst case for Bloom filter is when all bits are 1. Then anything also "hit".
Pop Quiz 2	8/10	—	Could do with less imul instructions.
Pop Quiz 1	5/10	—	Your understanding of endianness is incorrect. Please clarify with friends or me.