**CPSC/ECE 3220 - Study Guide for Midterm Exam, Summer 2020**

**Coverage: OSPP textbook, chapters 1-4, except for sections 2.5 (x86 context switches) and 2.8 (upcalls)**

1. Be able to define and discuss the following terms:

operating system

resource

virtualization

protection

security

privacy

principle of least privilege

proprietary system

open system

batch operating system

time-sharing operating system

virtual machine

host operating system

guest operating system

overhead

efficiency

response time

throughput

utilization

predictability

reliability

availability

operating system kernel

monolithic kernel

microkernel

portability

application programming interface (API)

hardware abstraction layer (HAL)

dynamically loadable device driver

BIOS ROM

bootstrap loader (bootloader)

dual-mode execution

user execution mode

kernel execution mode

privileged instruction

hardware timer

program counter (PC)

processor status register (PSR)

stack pointer (SP)

interrupt (also known as an asynchronous or external interrupt)

system call interrupt (also known as a syscall instruction,

supervisor call instruction (SVC), software interrupt

instruction (SWI or INT), or trap instruction)

exception (also known as a fault or trap)

interrupt vector table

interrupt handler

return from interrupt instruction

interrupt masking (also known as interrupt permission)

nested interrupts

poll

polling

user stack

interrupt stack (per processor)

kernel stack (per thread)

heap

system call library routine

stub function (also know as a wrapper function)

time of check versus time of use (TOCTOU)

program

process

thread

multithreaded process

PCB (process control block)

TCB (thread control block)

thread metadata

green threads

scheduler activations

parent process

child process

POSIX fork()

POSIX exec()

POSIX wait()

producer/consumer design pattern

client/server design pattern

fork/join concurrency

thread create

thread yield

thread join

thread exit

thread context switch

voluntary thread context switch

involuntary thread context switch (e.g., hardware timer interrupt)

thread scheduler

thread state (e.g., init, ready, running, waiting, finished)

idle thread

processor sleep mode

ready list

waiting list

running list

finished list

asynchronous I/O

event-driven programming

continuation

2. Be able to perform the following:

a) Take a specific feature or policy of an OS and explain whether it represents the OS serving in the   
 role of referee, illusionist, or glue.

b) Contrast kernel and user mode in an operating system.

c) Explain the actions of hardware in response to an interrupt.

d) Discuss how user processes or threads invoke and interact with the OS; and, be able to explain why   
 a normal subroutine call is not appropriate. (OS invocation requires a change of execution mode

and as well as enforcing the entry point address.)

e) Explain why an OS is said to be interrupt driven.

f) Describe the differences between a program, a process, and a thread.

g) Discuss how a thread is created and how it appears before the first instruction is executed.

h) Explain the rationale for each thread having a user stack and a kernel stack.

i) Distinguish among thread states during the lifetime of a thread and explain reasons for thread state  
 changes.

j) Describe the data structures needed to support thread management and thread scheduling.

**OSPP textbook, chapters 5-6**

1. Be able to define and discuss the following terms:

concurrency

real concurrency

apparent concurrency

race condition

lost update problem

instruction reordering

memory barrier, a.k.a. fence instruction

synchronization

critical section

safety property

liveness property

starvation

busy waiting

mutual exclusion

lock

acquire lock

release lock

state variable

synchronization variable

condition variable

wait on condition variable

signal on condition variable

broadcast on condition variable

blocking bounded queue

readers/writers lock

synchronization barrier

disable and enable interrupts

atomic read-modify-write instruction

test-and-set instruction

spin lock

semaphore

signaling pattern

mutex pattern

multiplex pattern

fine-grain locking

false sharing

per-processor data structure

test and test-and-set

MCS lock

compare-and-swap

optimistic concurrency control

lock-free data structure

deadlock

starvation

bounded resources

no preemption

wait while holding, a.k.a. multiple independent requests

circular waiting

mutually recursive locking

lock ordering

nested waiting

Banker's algorithm

safe state (with regard to deadlock)

unsafe state (with regard to deadlock)

deadlocked state

2. Be able to perform the following:

a) Demonstrate the potential run-time problems arising from the concurrent operation of many separate   
 threads (e.g., construct and explain an example of the lost update problem).

b) Use a synchronization technique to control concurrency among multiple threads (e.g., implement   
 mutual exclusion using a lock; implement blocking bounded buffer using a lock and condition

variables).

c) Identify errors in lock and condition variable usage in example code segments (e.g., accessing a   
 condition variable outside a locked region).

e) Evaluate the possible performance problems of a proposed multi- object locking scheme (e.g., false   
 sharing).

f) Compare and contrast a readers/writers lock to mutual exclusion.

e) Compare and contrast RCU to a readers/writers lock.

g) Give an example of the use of compare-and-swap that avoids the ABA problem.

h) List the four necessary conditions for deadlock.

i) Contrast the three approaches to deadlock (i.e., prevention, avoidance, and detect and recover).