Referee / Illusionist / Glue. Circle **only one** of R, I, or G. (1 point each)

1. / / **G** Libraries.
2. / **I** / Virtual machine.
3. / **I** / Guest Operating System
4. **R** / / Enforcing file access permissions.
5. / / **G** Disk details such as sector size are hidden.
6. **R** / / Isolates running programs from one another.
7. / **I** / Able to support multiple users simultaneously.
8. / / **G** Represents the largest section of code in the O/S.
9. **R** / / Controls communications between users and processes.

10. (2 points) What is the difference, if any, between Reliability and Availability?

Reliability is the correct operation of a system; and Availability the fraction of time (0.0 to 1.0) the system is working.

11. (2 points) What is the difference, if any, between Efficiency and Overhead?

Efficiency and Overhead are competing duals. Overhead is the cost (computation, memory, communication, storage, ...) of implementing an abstraction; and Efficiency is the lack of overhead.

12. (2 points) What is the difference, if any, between Security and Privacy?

Security is the protection of a computer's operations so that a malicious attacker cannot perform unauthorized operations. Privacy is the protection of the user data from unauthorized access.

13. (2 points) What is the difference, if any, Host Operating System and Guest Operating System?

A Host Operating System runs on the native hardware and provides a virtual machine abstraction. The Guest Operating System runs on a virtual machine.

14. (2 points) Older computer O/S' tended to be Batch systems (with 1 processor) and newer computer O/S' tend to be Interactive systems (with 2+ processors). Considering the users, what design feature has become more important in newer O/S'?

Response Time.

The Batch O/S focus on throughput and the Interactive O/S focus on Response Time. Response Time is the time the users wait for a task to complete, irrespective of whether the task is running or waiting.

15. (2points) Explain why an O/S supports communication through a file system.

File storage is a way to store data for later use (time insensitive), programs can run at different times and still communicate.

16. (2 points) Explain why an O/S supports communication through messages (pipes/sockets) passed between applications.

Most communication between applications occur through messages and is an effective way to commutate in real-time between programs.

17. (2points) Explain why an O/S supports communication through shared regions of memory.

Memory sharing is useful for large amounts of data (complex data structures) to prevent duplication.

18. (2 points) MTTR, and therefore availability, can be improved by reducing the time to reboot a system after a failure. What techniques might you use to speed up booting? Would your techniques always work after a failure?

Make a checkpoint at a regular internal as the O/S is running. After a "failure" restore back to the checkpoint when the system is restarted.

19. (2 points) When a user process is interrupted or causes a processor exception, the x86 hardware switches the stack pointer to a kernel stack, before saving the current process state. Explain why.

The user stack pointer may be corrupted. Switching to the kernel stack ensures that heir is a valid memory region to store the process state.
The kernel stack is inaccessible to the user and is protected. The kernel is able to store state information safely.

20. (2 points) For the "Hello World!" program, we mention that the kernel must copy the string from the user program to screen memory. Why must the screen's buffer memory be protected?

A malicious/flawed application could insert bad code into the protected memory. The O/S could then execute this bad code causing a system crash or a malicious app could gain control of the O/S.

21. (2 points) Explain the steps that an operating system goes through when the CPU receives an interrupt.

Several steps occur simultaneous (atomically):

Save the PC, IR, and PSR (Save the process state)

Switch to Kernel Mode

Disable/Defer future interrupts.

Load new PC from the Interrupt Vector Table

22. There are three mechanisms that are needed in dual-mode operation, privileged instructions, memory protection, and timer interrupts.

a. (2 points) Explain what could fail if privileged instructions are not enforced?

Process could directly perfrom I/O operation that could read/write anyone's file on disk

a. (2 points) Explain what could fail if memory protection is not enforced?

Process could access other data structures in the kernel and gain information that should be kept secured.

c. (2 points) Explain what could fail if timer interrupts are disabled?

Process could run in an infinite loop and never yield the processor back to the kernel.

23. (2 points) How does an O/S keep a user application from directly access hardware without controls?

The O/S implements protection through Dual-Mode Operation: Kernel/User.

Instructions that access hardware directly are considered privileged instructions.

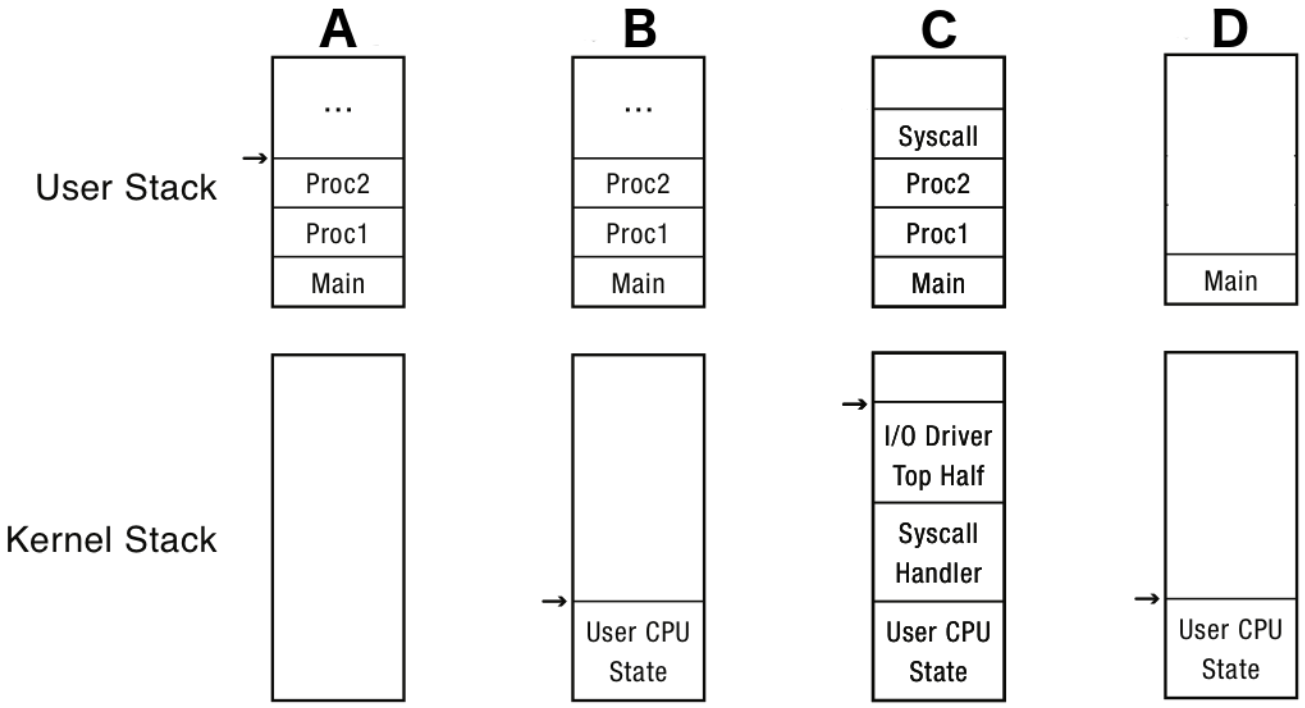
User Mode cannot execute privileged instruction.

24. (2 points) Describe the three types of user-mode to kernel-mode transfers.

System calls,
Interrupts, and
Exceptions

25. (4 points) Interrupt Stacks

For each description of a process, match the matching stack diagram.



__**D**__ New Process (Ready to run on Main)

__**A**__ Running Process

__**B**__ Ready Process (on Proc2)

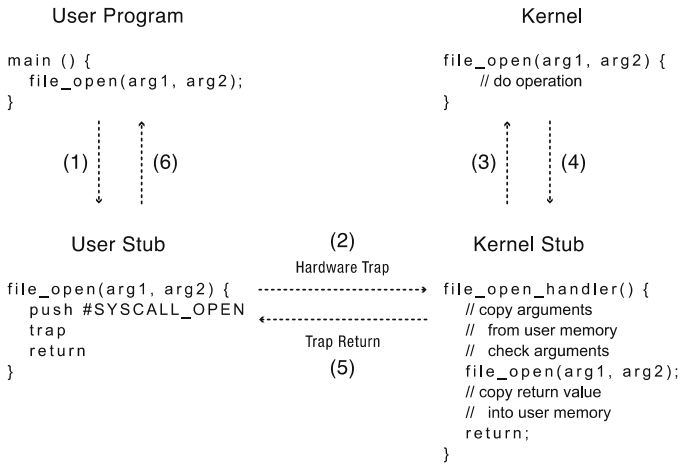
__**C**__ Waiting Process (on Proc2)

26. (4 points) What three tasks does the kernel stub need to preform before completing the system call (after step 2 but before step 3)

Locate system calls arguments

Validate parameters

Copy before check



27. (2 points) Can UNIX/LINUX fork() return an error? Why or why not?

Yes, the system runs out of resources (memory, Thread Control Blocks(TCB), etc.)

28. (2 points) Can UNIX/LINUX exec() return an error? Why or why not?

Yes, the executable doesn't exist or the user doesn't have access to the executable.

29. (2 points) What happens if we run the following program in UNIX/LINUX?

```
main() {
    while (fork() >= 0) ;
}
```

The maximum process will be created and no other process will be created. It will lock up the O/S. As one process dies another will be created.

30. (2 points) Given the code to the right;
Explain what must happen for the UNIX wait (line 8) to return immediately and successfully.

The child process of the fork has run to completion before the wait system call is made

```
1 main() {
2   int child_pid = fork();
3   if (child_pid == 0) {
4     printf ("I am process #%d\n", getpid());
5     return 0;
6   } else {
7     printf ("I am process #%d\n", getpid());
8     wait(child_pid);
9     printf ("I am the parent of process #%d\n", child_pid);
10    return 0;
11  }
12 }
```

31. (3 points) Given the code to the right,
How many different copies of the variable x are there?

3

What are their values where their process finishes?

Any order
{20, 15, 10}

```
1 main() {
2   int child = fork();
3   int x = 5;
4   if (child == 0) {
5     x += 5;
6   } else {
7     child = fork();
8     x += 10
9     if (child) {
10      x += 5;
11    }
12  }
13 }
```

32. (2 points) UNIX/LINUX uses one system command "open(args)" for I/O.
Why does Unix NOT use "open/create/exists" for I/O?

The UNIX/LINUX system call open(args) is an all-purpose atomic instruction. The arguments to the system call control the behavior (opening,creating,appending,reseting,...) of the I/O. A single system call simplifies the interface to the user apps.

33. (4 points) All operation on UNIX/LINUX I/O uses the same set of system calls.
List four of them:

open()
close()
read()
write()

34. (2 points) How is a Microkernel "better" than a Monolithic kernel?

Smaller core kernel
More modular libraries
Easier O/S updates
Simpler code base

35. (2 points) How is a Microkernel "poorer" than a Monolithic kernel?

More system calls needed for communication needed between user libraries and kernel code.

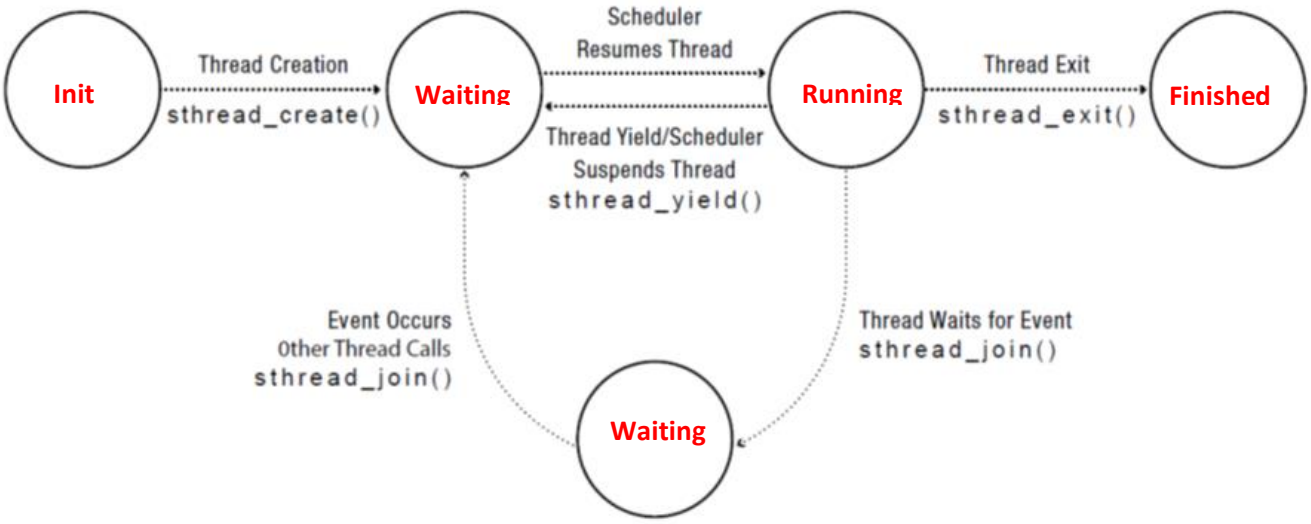
36. (2 points) Can a UNIX/LINUX pipe() be used for full-duplex communication? Why or why not?

No, a pipe can only support half-duplex communication, information can only flow one way through a pipe.

37. (1 point) Which UNIX/LINUX system call creates a new process?
exec() or fork()

fork() is the system call that creates a new process by "cloning" the currently running process. exec() is used to run a different process (but does not create a new process)

38.(4 points) In the following thread state diagram, label the states (i.e., write the labels inside the circles) with the state names: Finished, Init, Ready, Running, Waiting.



39. (2 points) A shared counter starts with value 0. What is the set of possible final values for the counter when the following three threads are run without mutual exclusion?

shared_counter = 0;

int local;	int local;	int local;
T1S1: local = shared_counter;	T2S1: local = shared_counter;	T3S1: local = shared_counter;
T1S2: local = local + 1;	T2S2: local = local + 1;	T3S2: local = local + 1;
T1S3: shared_counter = local;	T2S3: shared_counter = local;	T3S3: shared_counter = local;

In this example, a thread can be interrupted and result in a lost update. One or Two updates can be lost. The set of possible final values is {1, 2, 3}

40. (2 points) Suppose that you mistakenly create a local variable *v* in one thread *t1* and pass a pointer to v to another thread *t2*. Is it possible that a write by *t2* to *v* will cause *t1* to execute the wrong code?

Yes, Threads provide no protection from read/write by other threads. If a pointer/reference is passed from one thread, t1, to another thread, t2, within the same process, then the receiving thread, t2, can write bad code into the sending thread, t1.

threadHello.c program

```
1 #define NTHREADS 10
2 thread_t threads[NTHREADS];
3 main() {
4    for (i = 0; i < NTHREADS; i++) { thread_create(&threads[i], &go, i); }
5    for (i = 0; i < NTHREADS; i++) {
6       exitValue = thread_join(threads[i]);
7       printf("Thread %d returned with %ld\n", i, exitValue);
8    }
9    printf("Main thread done.\n");
10 }
11 void go (int n) {
12    printf("Hello from thread %d\n", n);
13    thread_exit(100 + n);
14 }
```

41. (2 points) For the threadHello program (above): What is the minimum and maximum number of times that the main thread enters the WAITING state (line 6)?

The minimum number of times is zero. All the children threads can all be created, and completed before the main thread reaches the call to thread_join. In this case the main thread would never wait.

The maximum number of times is ten. No child thread is finished before the main thread reaches the call to thread_join.

42. (2 points) For the threadHello program (above): Where is the parameter n (line 11) of the thread stored?

It is stored in the Thread Control Block (TCB) because it is a per-thread variable

43. (2 points) For the threadHello program (above): If the second for loop (lines 5-8) were deleted, what are the possible outputs of the program now? Why?

The possible outputs are "Main thread done" plus between 0 and 10 "Hello from thread i" (the prints can be in any order). When the main thread ends it exits and returns the Process Control Block (PCB). When the PCB is returned all the Thread Control Blocks (TCB) are returned at the same time (even if they are not done).

44. (2 points) Why do threads have "variable" speed?

A thread has no control over when it runs or doesn't run. The scheduler will schedule the threads and all threads will get CPU time, but the order is not predicatable.

45. (2 points) Describe what information is needed in a Thread Control Block (TCB)?

Copy of process registers
A stack

46. (2 points) What information is shared between threads of the same process?

Code
Data (Global variables)
Heap

True/False. Circle **only one** of T or F. (1 point each)

47. / **F** An interrupt creates a new thread.
48. / **F** An interrupt creates a new process.
49. **T** / An OS kernel can use internal threads.
50. / **F** On modern processors, all instructions are atomic
51. **T** / A Timer Interrupt is an example of an asynchronous interrupt.
52. / **F** Threads are more expensive for an O/S to create than processes.
53. / **F** A system call (e.g. open()) is an example of an asynchronous interrupt.
54. / **F** An OS should never create more processes than the available number of processors.
55. **T** / A loadable device driver means that the kernel does not have to be recompiled to use the device.
56. **T** / When a user attempts to execute a privileged instruction (while in user mode), the O/S should stop the process.