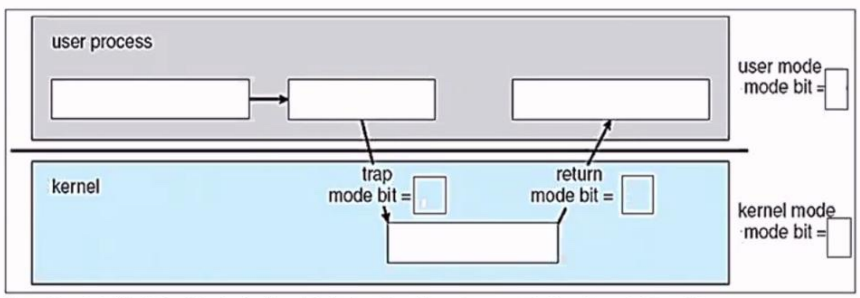
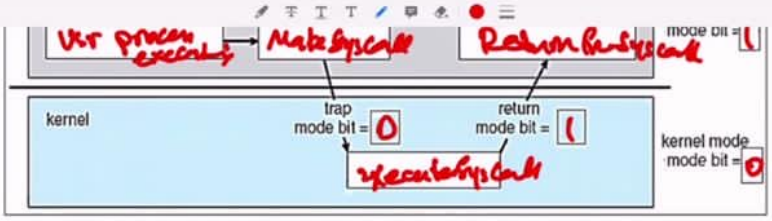
Question 1: In order to ensure the proper execution of the system, the operating system uses two modes of execution: user and kernel. Using the figure below and the provided list of steps, show the correct order of steps performed when a user process requests a service from the operating system. You need to clearly write each step into the correct box in the figure, and also write the value of the mode bit in each of the respective boxes. [2 points, 0.25 each]

Steps: *Return from system call- User process executing- Make a system call- Execute system call.*



Question 2: Fill ***all*** the blanks in the table below by either (✔ or ✘). [2 points, 0.25 each]

Storage	Volatile (temporary)? ✔ or ✘	Random access device? ✔ or ✘
Flash		
Hard disk drive		
Optical disk		
Solid-state disk (SSD)		



Question 2: Fill ***all*** the blanks in the table below by either (✔ or ✘). [2 points, 0.25 each]

Storage	Volatile (temporary)? ✔ or ✘	Random access device? ✔ or ✘
Flash		
Hard disk drive		
Optical disk		
Solid-state disk (SSD)		

Question 3: Match each of the following operating system structures with their specifications [3 points, 0.5 each]

a) Monolithic b) Layered c) Microkernel d) Modular

1) Kernel provides core services, while other operating system services are implemented as dynamically loadable modules that communicate freely via interfaces.	
2) Except for the minimal core services, most services run in user space communicating via message passing.	
3) System is divided into a number of distinct layers, starting from the hardware layer	

s almuhaideb's screen

Optical disk			
Solid-state disk (SSD)			

Question 3: Match each of the following operating system structures with their specifications [3 points, 0.5 each]

a) Monolithic b) Layered c) Microkernel d) Modular

1) Kernel provides core services, while other operating system services are implemented as dynamically loadable modules that communicate freely via interfaces.	Modular
2) Except for the minimal core services, most services run in user space communicating via message passing.	Micro
3) System is divided into a number of discrete layers, starting from the hardware layer and ending with a user interface layer. Each layer may only invoke services from lower-level layers.	layered
4) Has no structure. All functionality available in a single binary file.	Monolithic
5) Despite their simplicity, these are difficult to implement, debug, or extend since they are tightly-coupled. However, they are fast and efficient since all communication is within the kernel space.	Monolithic
6) These are easy to implement and debug since they are loosely-coupled. However, defining the functionality of each layer is a challenging task.	layer

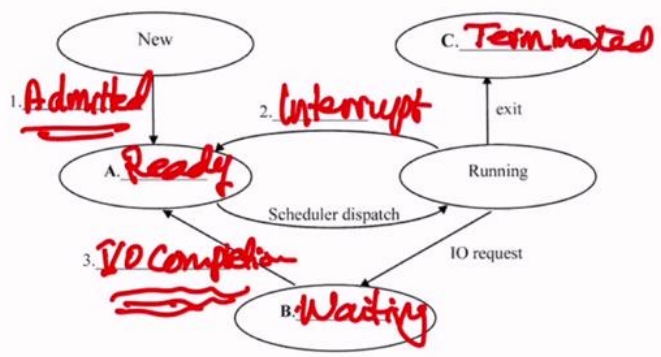
sequential access magnetic tape

s almuhaideb's screen

9:33 AM Thu 12 Nov

Done

Question 4: Suppose the following diagram of process state. Identify the **states** of a process in A, B and C blanks that can go through during its life cycle and the possible **actions** (events) in 1, 2,3 blanks that cause a process to change its states. [1.5 points, 0.25 each]



Question 5: [4.5 points]

Part A: Given the code below and assuming ***fork()*** never fails. Answer the following questions. [2 points]

1. How many "Hello" lines are printed?

9:35 AM Thu 12 Nov

Done

Question 5: [4.5 points]

Part A: Given the code below and assuming ***fork( )*** never fails. Answer the following questions. [2 points]

1. How many "Hello" lines are printed?

   4

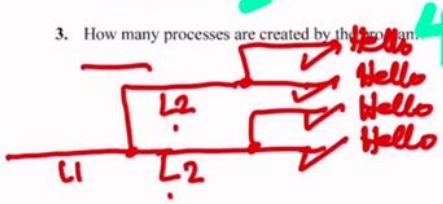
2. How many L1 and L2 are printed? [0.5 each]

   L1: 1

   L2: 2

3. How many processes are created by the program? 4

```
int main() {
    printf("L1\n");
    fork();
    printf("L2\n");
    sleep (10);
    fork ();
    printf ("Hello\n");
return 0;}
```



s almuhaideb's screen

9:35 AM Thu 12 Nov

Done

Part B: Given the code below and assuming ***fork( )*** and ***execlp( )*** never fail. Suppose the **parent** process id and **child** process id is **1500** and **1600**, respectively. [2.5 points, 0.5 each]

Parent Child

```
int main() {
  pid_t pid;
  pid = fork();
  int num = 5;
  if  (pid == 0) {
      printf("%d", getppid());..................... Line A
      execlp("date","date", NULL);
      num *= 6;
  } else if (pid > 0) {
      wait(NULL);
      num *= 2;
      printf ("%d",getpid());..................... Line B
 }
 printf ("%d", getpid()); ..................... Line C
 printf ("%d", num); ..................... Line D
return 0;}
```

1. What value did the *fork* function return to the child process?

9:53 AM Thu 12 Nov

Done

Question 6: [5 points]

Part A: Determine if the following problems exhibit **Task Parallelism** or **Data Parallelism**. [1 point, 0.5 each]

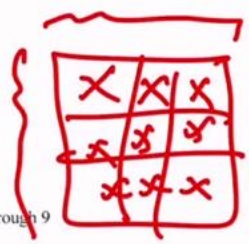
Problem 1: This program will be passed a series of numbers on the command line and will then create three separate worker threads to do the following:

- One thread will determine the average of the numbers,
- The second will determine the maximum value,
- And the third will determine the minimum value.

Answer: ..... Task

Problem 2: Given 9x9 matrix, threads are created to do the following criteria:

- A thread to check that each column contains the digits 1 through 9
- A thread to check that each row contains the digits 1 through 9
- Nine threads to check that each of the 3 × 3 sub grids contains the digits 1 through 9



Answer: ..... Task

s almuhaideb's screen

9:55 AM Thu 12 Nov

Done

Part B: Choose the word that best matches the description in the table: ***Process*** or ***Thread***. [2 points, 0.5 each]

Description	*Process* or *Thread*
It requires less time to be created.	T
It takes longer time to perform context-switch between them.	P
Should be made carefully since they are interdependent.	T
They are sharing the same global variables and address space	T

Part C: Indicate which of the following multi-threading models each limitation or advantage in the table pertains to: [2 points, 0.5 point each]

a) Many-to-one model b) One-to-one model c) Many-to-many model

Limitation	Multithreading model (a, b or c)
Multiple threads cannot run in parallel on multiprocessors.	
It's the most flexible model, but in practice it is difficult to	

Should be made carefully since they are interdependent.	T
They are sharing the same global variables and address space	T

Part C: Indicate which of the following multi-threading models each limitation or advantage in the table pertains to: [2 points, 0.5 point each]

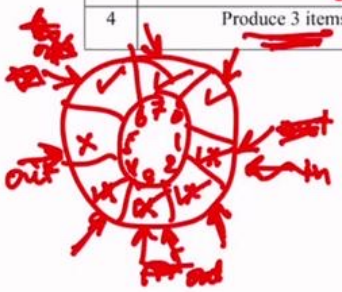
a) Many-to-one model b) One-to-one model c) Many-to-many model

Limitation	Multithreading model (a, b or c)
Multiple threads cannot run in parallel on multiprocessors.	Many to one
It's the most flexible model, but in practice it is difficult to implement.	Many to many
The entire process will block if a thread makes a blocking system call.	Many to one
Maps each user-level thread to one kernel thread, so may burden the performance of a system.	One to one.

9:45 AM Thu 12 Nov

Done

#	Action	in	out	Empty?	Full?
0	Initial Position	1	6	No	No
1	Consume 3 items	1	1	Yes	No
2	Produce 2 items	3	1	No	No
3	Consume 4 items 2left	3	3	Yes	No
4	Produce 3 items	6	5	No	No



9:47 AM Thu 12 Nov

Done

```
int result;
void thread_func( void *param );
int main( ) {
   int num;
   int pid, status;
   pthread_t thread1, thread2;
   result= 1000;
   pthread_create(&thread1, NULL, thread_func, NULL); _________ Line A
   pthread_create(&thread2, NULL, thread_func, NULL); _________ Line B
   pthread_join(thread1, NULL);
   pthread_join(thread2, NULL);
   num= 17; result= 17;
   pid=fork();
   if (pid == 0) { /* this is the child */
       num= 13; result= 23;
       printf("%d , %d /n", result, num);  _________ Line C
       exit(0);
   }
   else { /* this is parent */
        wait(&status);
        printf("%d , %d /n", result, num);  _________ Line D
```

s almuhaideb's screen

9:48 AM Thu 12 Nov

Done

```
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    num= 17; result= 17;
    pid=fork();
    if (pid == 0) { /* this is the child */
        num= 13; result= 23;
        printf("%d , %d /n", result, num);  _________ Line C
        exit(0);
    }
    else { /* this is parent */
         wait(&status);
         printf("%d , %d /n", result, num);  _________ Line D
         }
    exit(0);
}
void thread_func(void *param) {
     int tnum= 8;
     result++;
     printf("%d , %d /n", result, tnum*2);  _________ Line E
     pthread_exit(0);
     }
```

9:46 AM Thu 12 Nov

Done

```
        printf("%d , %d /n", result, num);  _________ Line D
        }
    exit(0);
}
void thread_func(void *param) {
     int tnum= 8;
     result++;
     printf("%d , %d /n", result, tnum*2); _________ Line E
     pthread_exit(0);
     }
```

9:52 AM Thu 12 Nov

Done

1. How many thread(s) will be created including the parent and child threads?

2. What will be printed in **Line E** due to the first ***pthread_create*** in **Line A**?

   1001 16

3. What will be printed in **Line E** due to the second ***pthread_create*** in **Line B**?

   1002 16

4. What will be printed in **Line C**?

   23 13

5. What will be printed in **Line D**?

9:40 AM Thu 12 Nov

Done

1. What value did the *fork* function return to the child process? 0

2. What will be printed due to ***getppid*** system call in **Line A**?

1500

3. What will be printed due to ***getpid*** system call in **Line B**?

1500

4. What will be printed due to ***getpid*** system call in **Line C**?

1500

5. Explain the circumstances under which **Line D** will be output the value of ***num*** for child process? [0.5 point]